

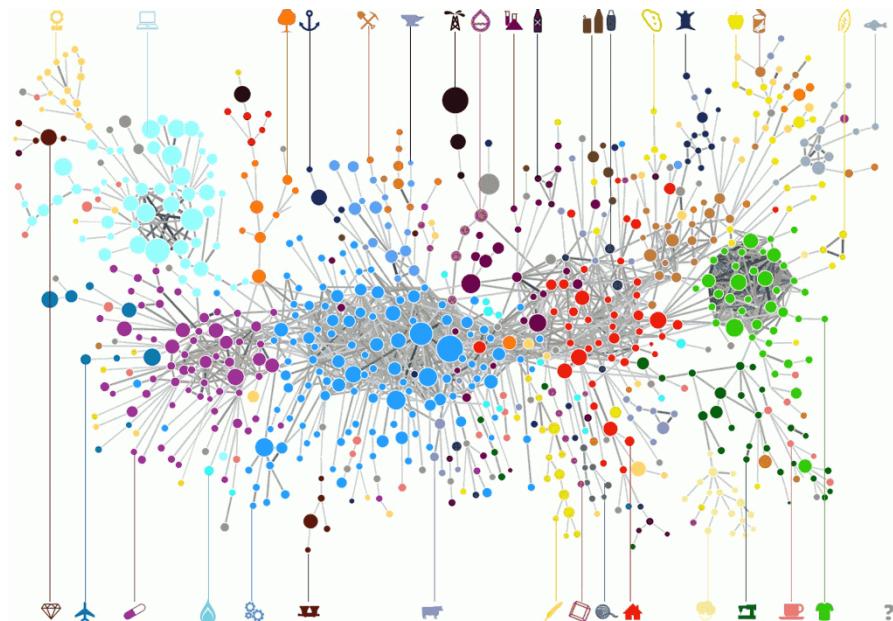
Département d’Informatique

Licence fondamentale : Sciences Mathématiques et Informatique (SMI)

Année universitaire : 2018-2019

Projet de Fin d’Etude

Détection de communautés dans les réseaux sociaux



Elaboré par :

Omar Mellal

Tarik Lemkadem

Encadré par :

Pr.Rachid Atay

Soutenu le 9 Juillet 2019 devant le jury composé de

M. ATAY, Professeur - Faculté des Sciences Meknès

M. BOURRAY, Professeur - Faculté des Sciences Meknès

M. OUANAN, Professeur - Faculté des Sciences Meknès

Faculté des Sciences, B.P. 11201 Zitoune,
Meknès, MAROC

Tél : 05 35 537 321 - Fax : 05 35 536 808

E-mail : doyen@fs-umi.ac.ma - Site web :

www.fs-umi.ac.ma

كلية العلوم، صندوق البريد : 11201 - الزيتون

مكناس، المغرب

الهاتف : 05 35 536 808 - الفاكس : 05 35 537 321

البريد الإلكتروني : doyen@fs-umi.ac.ma - الموقع الإلكتروني : doyen@fs-umi.ac.ma

Chapitre –

Remerciements

Nous tenons à remercier en premier lieu notre Dieu qui nous a donné la santé, la force et le courage pour mener à bien l'étude et l'implémentation de ce projet.

Nous tenons à remercier notre encadrant Dr. Rachid ATAY pour son dévouement et la confiance qu'il nous a accordé pour la réalisation de ce travail. sa disponibilité, son enthousiasme et sa générosité ont été un grand soutien dans notre travail.

Nous tenons à remercier tous nos enseignants qui ont contribué à notre formation. Nous tenons enfin à remercier nos chères collègues et aussi toutes les personnes qui nous ont aidé de près ou de loin à compléter ce travail.

Table des matières

1	Introduction	11
1.1	La théorie des graphes	11
1.1.1	Éléments de théorie des graphes	11
1.2	Mise en contexte et problématique	14
1.3	But du PFE	15
2	Communautés dans un réseau social	16
2.1	Réseaux sociaux	16
2.1.1	Définition du réseau social	16
2.1.2	Analyse des réseaux sociaux (SNA)	17
2.1.3	Twitter	18
2.1.4	Les types de réseaux sociaux	18
2.1.5	Les structures des réseaux sociaux	19
2.2	Communautés et graphes	24
2.2.1	Définition de la communauté	25
2.2.2	Modularité	25
2.2.3	clustering	26
2.3	Les algorithmes de détection de communautés	27
2.3.1	Les approches de détection	27
2.3.2	Les Algorithmes étudiés	31
3	Graphes aléatoires	37
3.0.1	Définition du graphe aléatoire	37
3.0.2	Pourquoi les graphes aléatoires ?	37
3.1	Rappels mathématiques	37
3.1.1	Probabilité et ses lois	37
3.2	Modèle d'Erdős-Rényi	40
3.2.1	Définition générale	40
3.2.2	L'avantage du modèle Erdős-Rényi	41
3.2.3	Implémentation du modèle Erdős-Rényi	41
3.2.4	Complexité	42
3.3	Modèle de Barabási-Albert	42
3.3.1	Définition générale	42
3.3.2	L'avantage du modèle Barabási-Albert	43
3.3.3	Implémentation du modèle Barabási-Albert	43
3.3.4	Complexité	44
3.4	Modèle de Watts-Strogatz	44
3.4.1	Définition générale	44
3.4.2	L'avantage du modèle Watts-Strogatz	44
3.4.3	Implémentation du modèle Watts-Strogatz	45
3.4.4	Complexité	45

Chapitre – TABLE DES MATIÈRES

4 Application	46
4.1 Détection des communautés	46
4.1.1 Notre implementation	46
4.2 Générer un graphe réel	54
4.2.1 UCL 100	55
4.2.2 UCL 500	56
4.2.3 UCL 1000	57
4.2.4 UCL 4000	58
4.3 Comparaison des caractéristiques	58
4.3.1 Module de Erdos-renyi base sur les graphes de terrain :	59
4.3.2 Module de Barabasi-albert base sur les graphes de terrain :	60
4.3.3 Module de Watts-strogatz base sur les graphes de terrain :	61

Chapitre – TABLE DES MATIÈRES

Table des figures

2.1-1 Représentation graphique d'un réseau social	17
2.1-2 Représentation graphique des différents types de réseaux	20
2.1-3 Modèle Watts-Strogatz : D'un réseau régulier vers un réseau aléatoire	22
2.1-4 Comparatif des différentes structures et mesures associées Source : Borner et al. [27]	24
2.3-1 Exemple de réseau à 16 sommets divisé en 2 communautés avec Walktrap	32
2.3-2 Illustration de l'algorithme de Louvain	34
2.3-3 l'exécution de l'algorithme fast greedy sur Le réseau social du club de karaté de Zachary	36
3.2-1 Trois graphes aléatoires d'Erdös-Rényi	41
3.2-2 Le résultat de la fonction "Erdos_renyi_graph(n,p,seed=None,directed=False)" Pour n=50,p=0.5	42
3.3-1 Le résultat de la fonction "barabasi_albert_graph(40,15)" pour 40 noeuds	44
3.4-1 Le résultat de la fonction "G= nx.watts_strogatz_graph(100,10,0.2)" pour 100 noeuds	45
4.2-1 les graphes de UCL 100	55
4.2-2 les graphes de UCL 500	56
4.2-3 les graphes de UCL 1000	57
4.2-4 les graphes de UCL 4000	58

Chapitre – TABLE DES FIGURES

Table d’abréviations

E-R	Erdos-Renyi
E-B	Edge betweenness
GN	Girvan Newman
API	Application programming interface
PCC	plus courte chemine
G-A	Graphe Aléatoire
B-A	Barabasi-Albert
W-S	Watts-Strogatz
UCL	UEFA Champions League
UEFA	Union of European Football Associations

Chapitre – TABLE DES FIGURES

1 Introduction

Ce chapitre nous permet de donner un certain nombre de concepts qui seront utilisés par la suite. Tout d'abord la section 1-1 retrace l'historique de l'étude des graphes : de la théorie des graphes aux travaux actuels sur les grands graphes. Nous présentons en section 1-1-1 l'ensemble des notations et concepts de la théorie des graphes qui seront utiles dans les sections et chapitres suivants puis nous décrirons en 1-2 la mise en contexte et la problématique et Nous finissons par le but du PFE .

1.1 La théorie des graphes

Dans cette section, nous rappelons les notions essentielles. nous introduisons d'abord les concepts fondamentaux de la théorie des graphes,utilisés ici pour représenter et manipuler les réseaux sociaux. nous présentons ensuite la mise en contexte pour situer le sujet . Ainsi que le but de ce projet

1.1.1 Éléments de théorie des graphes

1. Définition d'un graphe :

Un graphe $G = (S, A)$ est un ensemble de noeuds et un ensemble $A \subseteq S \times S$ d'arêtes reliant ces noeuds.

2. Représentation d'un graphe :

— Listes d'adjacences

Soit le graphe $G = (S, A)$. On suppose que les sommets de S sont numérotés de 1 à n , avec $n = |S|$.

La représentation par listes d'adjacence de G consiste en un tableau T de n listes, une pour chaque sommet de S . Pour chaque sommet $s_i \in S$, la liste d'adjacence $T[s_i]$ est une liste chaînée de tous les sommets s_j tels qu'il existe un arc ou une arête $(s_i, s_j) \in A$. Autrement dit, $T[s_i]$ contient la liste de tous les sommets successeurs de s_i . Les sommets de chaque liste d'adjacence sont généralement chaînés selon un ordre arbitraire.

Si le graphe est valué (par exemple, si les arêtes représentent des distances), on peut stocker dans les listes d'adjacence, en plus du numéro de sommet, la valuation de l'arête.

Dans le cas de graphes non orientés, pour chaque arête (s_i, s_j) , on aura s_j qui appartiendra à la liste chaînée de $T[s_i]$, et aussi si qui appartiendra à la liste chaînée de $T[s_j]$.

— Matrice d'adjacence

Soit le graphe $G = (S, A)$. On suppose que les sommets de S sont numérotés de 1 à n , avec $n = |S|$. La représentation par matrice d'adjacence de G consiste en une matrice booléenne M de taille $n * n$ telle que $M[i][j] = 1$ si $(i, j) \in A$, et $M[i][j] = 0$ sinon. Si le graphe est valué (par exemple, si des distances sont associées aux arcs), on peut utiliser une matrice d'entiers, de telle sorte que $M[i][j]$ soit égal à la valuation de l'arc (i, j) si $(i, j) \in A$. S'il n'existe pas d'arc entre 2 sommets i et j , on peut placer une valeur particulière (par exemple 0 ou $-\infty$ ou null) dans $M[i][j]$. Dans le cas de graphes non orientés, la matrice est symétrique par rapport à sa diagonale descendante. Dans ce cas, on peut ne mémoriser que la composante triangulaire supérieure de la matrice d'adjacence.

* Quelques remarques sur la complexité de ces deux représentations

-Matrice d'adjacence :

la complexité en espace : $O(|V|^2)$: potentiellement très mauvais

Pour accéder à un sommet : $O(1)$: optimal

Parcourir tous les sommets : $\Theta(|V|^2)$: potentiellement très mauvais

-Liste d'adjacence :

la complexité en espace : $O(|V|+|E|)$: optimal

Pour accéder à un sommet : $O(1)$: optimal

Parcourir tous les sommets : $\Theta(|V|)$: optimal

mais en tout cas si le graphe est dense c'est mieux d'utiliser la matrice d'adjacence

3. Types de graphes

— Les graphes non orientés :

Un graphe non orienté G est la donnée d'un couple $G = (S, A)$ tel que :

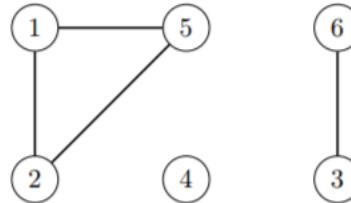
- S est un ensemble fini de sommets,

- A est un ensemble de couples non ordonnés de sommets $(s_i, s_j) \in S^2$

Une paire (s_i, s_j) est appelée une arête, et est représentée graphiquement par $s_i s_j$. On dit que les sommets s_i et s_j sont adjacents. L'ensemble des sommets adjacents au sommet $s_i \in S$ est noté

$\text{Adj}(s_i) = [s_j \in S, (s_i, s_j) \in A]$.

Par exemple,



représente le graphe non orienté $G = (S, A)$ avec $S = \{1, 2, 3, 4, 5, 6\}$ et $A = \{\{1, 2\}, \{1, 5\}, \{5, 2\}, \{3, 6\}\}$.

— Les graphes orientés :

Un graphe orienté G est la donnée d'un couple $G = (S, A)$ tel que :

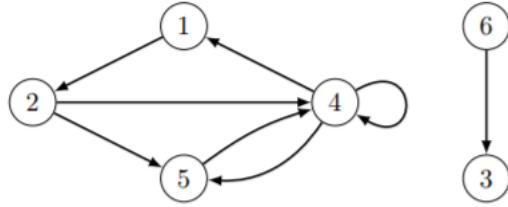
- S est un ensemble fini de sommets,

- A est un ensemble de couples ordonnés de sommets $(s_i, s_j) \in S^2$

Un couple (s_i, s_j) est appelé un arc, et est représenté graphiquement par $s_i \rightarrow s_j$, si s_i est le sommet initial ou origine, et s_j le sommet terminal ou extrémité. L'arc $a = (s_i, s_j)$ est dit sortant en s_i et incident en s_j , et s_j est un successeur de s_i , tandis que s_i est un prédécesseur de s_j . L'ensemble des successeurs d'un sommet $s_i \in S$ est noté

$\text{Succ}(s_i) = \{s_j \in S, (s_i, s_j) \in A\}$. L'ensemble des prédécesseurs d'un sommet $s_i \in S$ est noté $\text{Pred}(s_i) = \{s_j \in S, (s_j, s_i) \in A\}$.

Par exemple



représente le graphe orienté $G = (S, A)$ avec $S=\{1,2,3,4,5,6\}$ et $A=\{(1,2),(2,4),(2,5),(4,1),(4,4),(4,5),(5,4),(6,3)\}$.

4. Graphe simple :

Un graphe est dit simple lorsque, pour l'ensemble des sommets du graphe, il y a au plus une arête entre deux sommets distincts et pas de boucle.

5. Graphe pondéré :

Un graphe est pondéré si les arêtes sont annotées par des poids

- Exemple :réseau entre villes avec comme poids la distance entre les villes, réseau internet avec comme poids la bande passante entre routeurs, etc

6. Degré d'un graphe :

Le degré d'un graphe est le nombre maximal d'arêtes incidentes à tout sommet.

7. Densité d'un graphe :

La densité D d'un graphe est la probabilité que deux noeuds pris au hasard soient voisins :

$D= 2m/n(n-1)$.avec $m=$ nombre d'arêtes et $n=$ nombre de sommets .

8. Diamètre d'un graphe :

On appelle diamètre d'un graphe G , la distance maximale entre deux sommets du graphe G .

9. Parcours et marches aléatoires :

Un parcours de graphe est un chemin visitant chaque noeud du graphe. Un algorithme de parcours de graphe visite donc séquentiellement tous les noeuds du graphe.

il existe plusieurs algorithmes de parcours ,mais les plus utilisable sont parcours en largeur BFS (Breadth First Search) et parcours en profondeur DFS (Depth-First Search) dans notre pfe on va utiliser BFS pour calculer le diamètre .. ou parcourir un graphe

-Parcours en largeur(BFS) :



Python

```
def bfs(G,s) :
    couleur=dict()
    for x in G : couleur[x]='blanc'
    P=dict()
    P[s]=None
    couleur[s]='gris'
    Q=[s]
    while Q :
        u=Q[0]
        for v in G[u] :
            if couleur[v]=='blanc' :
                P[v]=u
                couleur[v]='gris'
                Q.append(v)
        Q.pop(0)
        couleur[u]='noir'
    return P
```

Complexité : Prends un temps en $O(n + m)$ pour un graphe avec n sommets et m arêtes

Plusieurs algorithmes de détection de communautés utilisent des marches aléatoires uniformes, un processus discret sur le graphe. À chaque instant, un marcheur est situé sur un noeud du graphe. Il se déplace à l'instant suivant uniformément aléatoirement vers l'un des voisins du noeud sur lequel il est situé. Le chemin formé est appelé une marche aléatoire.

1.2 Mise en contexte et problématique

Soit G un graphe non orienté et non pondéré. G est défini par $G = (V, E)$, où V représente l'ensemble des nœuds et E correspond à l'ensemble des liens qui relient les différents nœuds de G . Le but de la détection de communauté est de trouver une partition $P = C_1, \dots, C_k$ de k communautés de l'ensemble des nœuds de V . Chaque communauté C_i représente un sous-groupe de nœuds qui sont fortement connectés plus qu'ailleurs dans le graphe G . Il convient de noter que la valeur de k est inconnue et doit être identifiée automatiquement.

Les problèmes majeurs dans le processus de la détection des communautés consistent principalement à trouver une définition exacte d'une communauté, il n'existe pas une définition exacte d'une communauté mais plusieurs recherches ont donné des approches pour cette notion. Dans notre PFE nous allons utiliser la maximisation de modularité qui proposé par Girvan-Newman, cette approche donne une très bonne définition sur les communautés s'il n'existe pas de chevauchement. Après, pour détecter une communauté

dans un réseau social il est nécessaire de le modéliser par un graphe aléatoire, un autre problème se pose qui est la qualité de module proposer, pour les graphes aléatoires ils existent plusieurs modules plus au moins efficaces pour un réseau complexe et plusieurs caractéristiques à respecter

1.3 But du PFE

La détection de communauté est une approche importante pour mieux comprendre la structure du réseau complexe. Ces réseaux peuvent être biologiques, sociaux, géographiques ou technologiques. Une information formidable est présente dans ces réseaux. Si ceux-ci peuvent être exploités avec succès, de nombreux résultats importants peuvent être déduits. De nombreux algorithmes sont développés pour la détection de communautés. Mais la définition même de la communauté est discutable. Donc, il y a plusieurs problèmes et les défis à relever pour les recherches futures dans ce domaine. Dans ce projet, nous nous sommes principalement concentrés sur un problème majeur lié à la détection de communautés et sa modélisation par un graphe aléatoire et cela permet de faire des prédictions sur l'étude d'un réseau social

2 Communautés dans un réseau social

Dans ce chapitre, nous présentons d'abord le contexte des réseaux sociaux en général, dans la section 2.2. Nous introduisons également les concepts de détection des communautés qui seront utiles par la suite. Nous aborderons les méthodes de comparaison dans la section 2.3. Nous terminerons ce chapitre par les notions des communautés et graphes aléatoires qui sont l'introduction du prochain chapitre dans lequel on va traiter les graphes aléatoires et leurs propriétés .

2.1 Réseaux sociaux

Depuis 2002, date de la création de MySpace, qui devint le premier site social mondial, les sites sociaux ont envahi notre vie : Facebook, créé en 2004, a atteint 2,32 milliards d'utilisateurs mensuels en 2018 LinkedIn, créé en 2003, en a aujourd'hui environ 400 millions et Twitter (2006) compte 302 millions d'utilisateurs actifs. En France, Skyrock, créé en 2002, atteignait 21 millions d'utilisateurs en juin 2008 avant d'être atteint par la croissance de Facebook. En Chine, Tencent a 600 millions d'utilisateurs actifs par mois sur WeChat et 843 millions sur QQ, Sina Weibo 300 millions.

2.1.1 Définition du réseau social

Un réseau social représente un système d'entités en interaction. On le modélisera comme un graphe $G = (S, A)$ où S est un ensemble d'entités (les sommets ou noeuds du graphe) et A est l'ensemble des arcs (ou connexions) représentant les interactions entre ces sommets. Un site social comme Facebook peut également être représenté de cette façon : un noeud est un membre de Facebook, et deux noeuds sont connectés s'ils sont amis L'analyse de ces données permet par exemple de construire un graphe montrant les interactions entre les utilisateurs de différents

pays : ici les liens sont d'autant plus foncés que le nombre d'échanges est important. La visualisation des réseaux sociaux est un domaine très actif sur lequel nous reviendrons plus loin. Les entités d'un réseau social peuvent être de toutes sortes : pages web, membres d'un site social, comptes bancaires, protéines... et les liens peuvent représenter des interactions variées entre ces entités : liens d'amitié sur Facebook, follower sur Twitter, hyperliens sur le web, correspondance e-mail ou appels téléphoniques

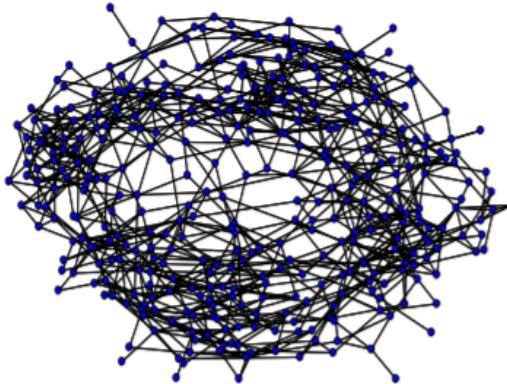


FIGURE 2.1-1 – Représentation graphique d'un réseau social

2.1.2 Analyse des réseaux sociaux (SNA)

L'analyse des réseaux sociaux est avant tout une boîte à outils permettant de visualiser et modéliser les relations sociales comme des nœuds (les individus, les organisations. . .) et des liens (relations entre ces nœuds). De ce fait, l'analyse des réseaux sociaux repose sur des visualisations graphiques issues d'algorithme permettant de calculer des degrés de force ou de densité entre les différents acteurs d'un réseau. Ainsi, l'analyse des réseaux sociaux est fondée sur une approche structurale des relations entre membres d'un milieu social organisé. Elle s'attache à décrire les interdépendances entre acteurs et permet une simplification de leur représentation que Lazega [1998, p. 6]¹ qualifie de « représentation simplifiée d'un système social complexe ». Cette simplification dans l'agencement des interdépendances est volontaire puisque l'analyse des réseaux sociaux se veut être une « technique d'exploration et de représentation » (Lazega, 1998, p.6). Par ailleurs, l'analyse des réseaux sociaux est intégrée à une réflexion plus vaste autour de la sociologie des groupes. La compréhension de la structure des ensembles sociaux repose sur l'étude des relations entre membres d'un milieu social

Tous les graphes qu'on va étudier sont extraits à partir de twitter à l'aide des outils comme Tweepy de python et Twitter streaming importé de Gephi

1. ANALYSE DES RESEAUX SOCIAUX ET COMMUNAUTES EN LIGNE : QUELLES APPLICATIONS EN MARKETING ?

2.1.3 Twitter

Twitter est un réseau social de microblogging géré par l'entreprise Twitter Inc. Il permet à un utilisateur d'envoyer gratuitement de brefs messages, appelés tweets, sur internet, par messagerie instantanée ou par SMS. Ces messages sont limités à 280 caractères.

Twitter a été créé le 21 mars 2006 par Jack Dorsey, Évan Williams, Biz Stone et Noah Glass, et lancé en juillet de la même année. Le service est rapidement devenu populaire, jusqu'à réunir plus de 500 millions d'utilisateurs dans le monde fin février 2012. Au 5 mars 2017, Twitter compte 313 millions d'utilisateurs actifs par mois avec 500 millions de tweets envoyés par jour et est disponible en plus de 40 langues.

API twitter

L'API Twitter est une passerelle ou interface de programmation permettant de se connecter aux données Twitter de façon automatisée.

L'API Twitter peut être utilisée pour extraire des données à des fins de veille sur les réseaux sociaux comme le font les plateformes de social listening. Dans ce dernier cas, on utilise la Search API de Twitter.

On a crée un compte développeur twitter dans le but de l'utiliser pour récupérer les données d'un tweet

2.1.4 Les types de réseaux sociaux

— Réseaux unipartis

Un réseau uniparti est un réseau qui ne contient que des liens connectant des noeuds d'un même type. Des exemples classiques sont les réseaux sociaux liant des individus entre eux, le réseau Internet liant un ensemble de routeurs, ou le WEB qui connecte un ensemble de sites internet. Un exemple de réseau uniparti peut être observé sur la Figure 2.2(a).

— Réseaux complets

Un réseau uniparti dans lequel tous les noeuds sont liés entre eux, c'est-à-dire qui possède $\frac{N \times (N-1)}{2}$ liaisons dans le cas d'un réseau non-orienté ou $N \times (N-1)$ liaisons dans le cas d'un réseau orienté, est appelé **réseau complet**. Un exemple de réseau complet peut être observé sur la Figure 2.2(c)

— Sous-réseaux

Un réseau $G' = (V', E')$ est dit **sous-réseau** de $G = (V, E)$ si $V' \subseteq V$ et $E' \subseteq E$, on note alors $G' \subseteq G$. Un exemple de sous-réseau du réseau de la Figure 2.2(a) peut être observé sur la Figure 2.2(d).

— Réseaux pondérés

Un réseau pondéré est un réseau dans lequel chaque lien $e = (v_i, v_j)$ est caractérisé par un poids w_{v_i, v_j} qui correspond à une valeur numérique affectée au lien. Évidemment, dans un réseau non-orienté si le lien $e = (v_i, v_j)$ appartient à E , nous avons $w_{v_i, v_j} = w_{v_j, v_i}$. Ce poids peut être soit calculé par des informations sur le graphe lui-même ou peut être obtenu à partir d'informations complémentaires. Par exemple, dans un réseau représentant les rencontres entre des individus, le poids peut être la fré-

quence de ces rencontres [Read 2008]. Nous montrons un exemple de réseau pondéré sur la Figure 2.2(e).

— **Réseaux bipartis**

Certains réseaux, comme les réseaux d'achats entre des consommateurs et les produits qu'ils achètent font intervenir deux types de noeuds ; on parle alors de réseaux bipartis. Plus formellement, un réseau est dit biparti s'il existe une partition de son ensemble de noeuds en deux sous-ensembles V_A et V_B telle que chaque lien du réseau ait une extrémité dans V_A et l'autre dans V_B . Un réseau biparti est représenté par un graphe $G = (V_A, V_B, E)$ où V_A et V_B représentent les deux ensembles indépendants et $E \subseteq V_A \times V_B$. Un exemple de réseau biparti peut être observé sur la Figure 2.2(f). Les groupes des noeuds $\{1,3,4\}$ et $\{2,5\}$ appartiennent respectivement aux ensembles V_A et V_B .

— **Composantes connexes**

Une **composante connexe** C d'un réseau G est définie comme un sous-réseau connecté de G . Deux composantes connexes $C_1 = (V_1, E_1)$ et $C_2 = (V_2, E_2)$ de G sont dites déconnectées s'il n'existe aucun chemin reliant un noeud v_i de V_1 à un noeud v_j de V_2 . La Figure 2.2(g) illustre un exemple de deux composantes déconnectées obtenues à partir du réseau de la Figure 2.2(a).

2.1.5 Les structures des réseaux sociaux

L'étude détaillée des propriétés structurelles de très grands réseaux a permis de mettre en évidence les structures topologiques particulières communes aux réseaux du monde réel. De nombreux travaux récents ont en effet montré qu'au-delà de leur différence sémantique, la plupart des réseaux issus du monde réel sont caractérisés par des propriétés topologiques analogues telles qu'une distribution des degrés qui suit une loi de puissance, une distance moyenne relativement faible ou la présence de communautés. Ces caractéristiques sont radicalement différentes des celles observées classiquement sur des réseaux réguliers ou aléatoires étudiés traditionnellement dans le domaine de la théorie des graphes.

Intéressons-nous aux quatre types des structures identifiés et aux modèles permettant leur génération.

— **Réseaux réguliers**

Les structures régulières sont les structures de réseau les plus simples, souvent utilisées dans des modèles d'automates cellulaires. Dans un réseau régulier, chaque noeud possède un nombre identique de liaisons. La densité du réseau est souvent faible, alors que le coefficient de clustering est, lui, relativement élevé. Il est d'ailleurs intéressant d'observer que comme la structure est régulière, le coefficient de clustering ne varie pas avec la taille du réseau. Enfin, la distance moyenne dans ces réseaux est souvent élevée, puisque la structure ne présente pas de "ponts" permettant de relier des individus fortement éloignés. Il y a plusieurs façons d'obtenir de tels réseaux. L'une des plus simples consiste à disposer les noeuds équitablement sur un cercle et à créer, pour chaque noeud, des connexions avec les x premiers noeuds situés à gauche et à droite de sa position. Naturellement, pour garantir une structure régulière, x doit être identique pour tous les noeuds v_i du réseau. Ainsi,

$$\forall v_i \in V, k_{v_i} = 2 \times x \quad (1)$$

Dans un tel réseau, la distribution des degrés est donc définie en un seul point $k = 2 \times x$, tel que $P(k) = 1$. Bien que ce type de structure s'observe en réalité très peu dans la nature, elle est en re-

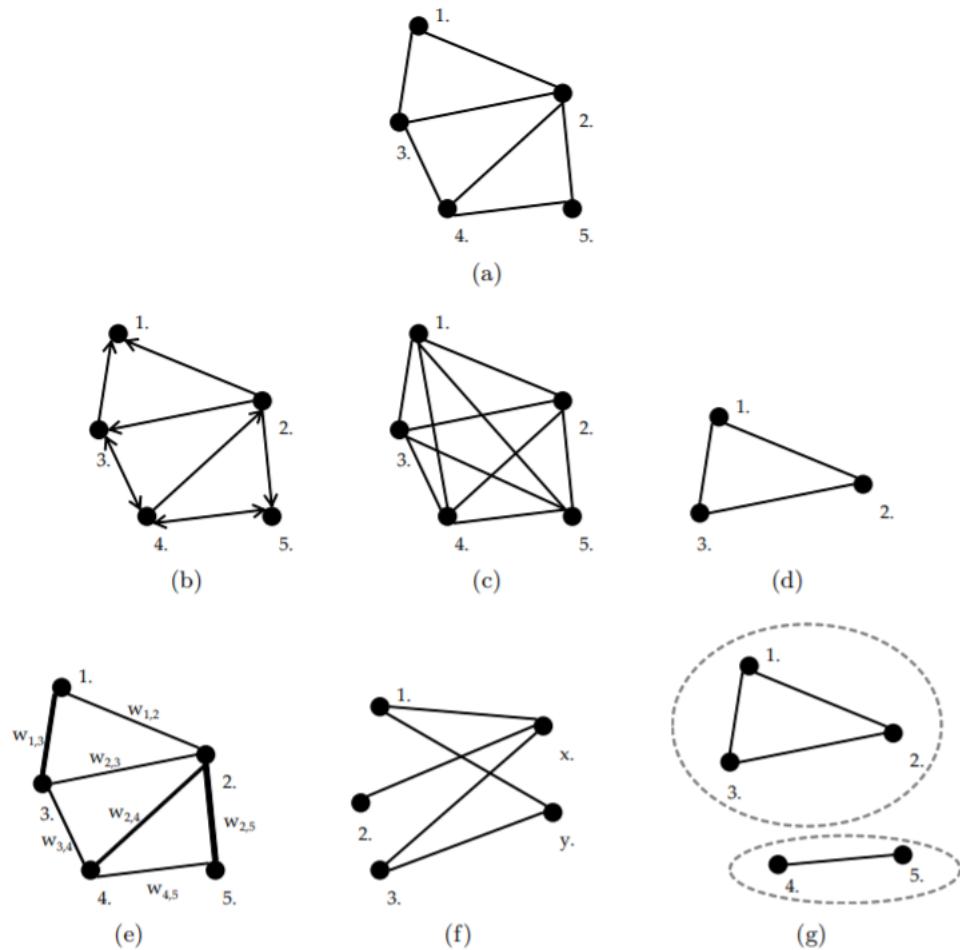


FIGURE 2.1-2 – Représentation graphique des différents types de réseaux

(a) Réseau non-orienté, (b) Réseau orienté, (c) Réseau complet, (d) Sous-réseau du réseau (a), (e) Réseau pondéré, (f) Réseau biparti, (g) Deux composantes connexes du réseau (a)

vanche souvent utilisée comme base pour la formation de réseaux plus réalistes.

— Réseaux aléatoires

Le terme de réseaux aléatoires fait référence à des structures au sein desquelles l'existence d'un lien entre deux noeuds est le résultat d'un processus aléatoire. Un réseau aléatoire se caractérise par une distribution des degrés dite "homogène", c'est-à-dire une loi de poisson en forme de cloche. Cela traduit le fait que dans le réseau, une forte proportion de noeuds est moyennement connectée, alors qu'une plus faible proportion est fortement et faiblement connectée. On observe également que les réseaux aléatoires ont des distances moyennes relativement faibles. Cela s'explique par le fait que quand des liens sont créés aléatoirement entre des noeuds, la probabilité qu'un noeud se retrouve isolé des autres est plutôt faible.

Le modèle de génération de réseaux aléatoires le plus connu est celui proposé par Erdos et Rényi [Erdos 1960]². Dans le modèle Erdos-Rényi, chaque lien potentiel du réseau est créé avec une probabilité p , indépendante de l'existence des autres liens. Autrement dit, chaque couple de noeuds (v_i, v_j) a une probabilité p d'exister dans le réseau.

Une variante de ce modèle consiste à choisir uniformément au hasard un réseau G dans l'ensemble de tous les réseaux possibles contenant N noeuds.

Bien que les réseaux aléatoires aient été l'objet de recherches intensives, ce type de structure ne reproduit pas totalement les caractéristiques observées dans les réseaux du monde réel. Barabasi et Bonabeau [Barabasi 2003]³ expliquent par exemple que "*malgré le placement aléatoire des liens, la plupart des noeuds ont environ le même nombre de connexions. En effet, dans un réseau aléatoire, les degrés suivent une distribution de Poisson avec une forme de cloche et il est extrêmement rare de trouver des noeuds qui ont, de manière significative, plus ou moins de liens que la moyenne*".

— Réseaux petit-monde

Un réseau petit-monde désigne à l'origine une structure dans laquelle les chemins entre deux noeuds quelconques sont généralement très courts, c'est-à-dire que la distance moyenne dans le réseau est relativement faible. Ce concept a notamment été mis en évidence par la célèbre expérience de Milgram [Milgram 1967]⁴ (détalée dans la section suivante), qui montra que le nombre d'intermédiaires nécessaires pour atteindre deux individus dans un réseau était d'environ 6. Aujourd'hui un réseau de type petit-monde est caractérisé par deux propriétés : une distance moyenne relativement faible dans le réseau et un coefficient de clustering élevé.

Les premiers travaux à s'intéresser à la génération de réseaux petit-monde sont ceux de Watts et Strogatz [Watts 1998]⁵, qui ont proposé un modèle de génération simple, connu sous le nom de modèle Watts-Strogatz et basé sur l'extension d'un réseau régulier. Comme illustrée sur la Figure 2.4, la procédure est la suivante. (1) Un réseau régulier est généré selon la méthode présentée précédemment. (2) Chaque lien subit ensuite un processus de réécriture selon une probabilité pr . La procédure de réécriture consiste à remplacer un lien (v_i, v_j) par (v_i, v_k) avec k choisi aléatoirement, tel que $k \neq i$ et $(v_i, v_k) \notin E$.

2. Erdős, P. ; Rényi, A. (1959). "On Random Graphs. I"

3. E. Ravasz and A-L Barabasi (2003) Hierarchical organization in complex networks.

4. Milgram, S. (1967) The Small World Problem. Psychology Today, 2, 60-67.

5. Watts, D. J., Strogatz, S. H. (1998). Collective dynamics of "small-world" networks. Nature.

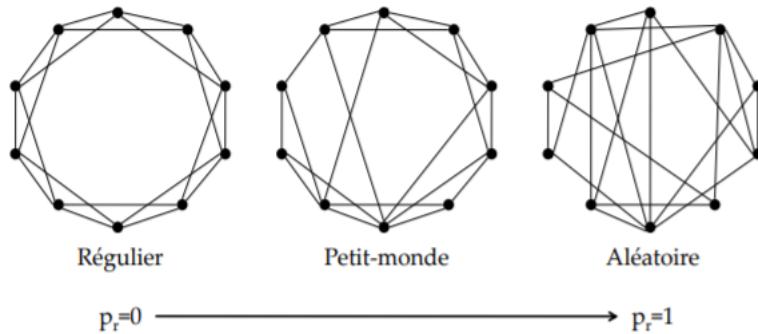


FIGURE 2.1-3 – Modèle Watts-Strogatz : D'un réseau régulier vers un réseau aléatoire

Comme nous l'avons expliqué précédemment, les réseaux réguliers présentent des coefficients de clustering et des distances moyennes relativement élevés. Ainsi, la réécriture aléatoire de quelques liens introduit des connexions entre des noeuds potentiellement situés sur de longues distances dans le réseau régulier initial, ce qui réduit considérablement la distance moyenne entre les noeuds. Quand la probabilité p_r reste relativement faible, de nombreux noeuds conservent leurs connexions avec leurs voisins initiaux, c'est-à-dire ceux du réseau régulier. Le coefficient de clustering global reste donc relativement élevé alors que la distance moyenne est, elle, réduite, permettant ainsi l'émergence de la propriété petitmonde. Watts et Strogatz ont également montré que plus p_r se rapproche de 1, plus le réseau tend vers une structure aléatoire, puisque tous les liens sont réécrits aléatoirement. La propriété petit-monde est aujourd'hui couramment observée dans de nombreux réseaux du monde réel. Toutefois, bien que le modèle de Watts-Strogatz permette de générer aisément des réseaux de type petit-monde, il a souvent été critiqué pour son incapacité à produire des noeuds possédant un degré élevé, une caractéristique souvent observée dans les réseaux du monde réel.

— Réseaux scale-free

Une autre découverte fondamentale dans le domaine des réseaux a été faite en 1999 par Barabasi et Albert [Barabasi 1999], alors qu'ils étudiaient une partie du réseau de pages WEB. Ils montrèrent en effet que contrairement aux réseaux aléatoires, le réseau étudié présentait une distribution des degrés hétérogène, dans laquelle seule une faible proportion de noeuds, appelés "hubs", avait beaucoup plus de connexions que les autres. De telles structures ont par la suite également été observées dans de nombreux autres réseaux tels que le réseau Internet, les réseaux de citations d'articles scientifiques et certains réseaux sociaux. Ainsi, un réseau scale-free est un réseau dont la distribution des degrés suit une loi de puissance. Cela se traduit par le fait que dans le réseau, une forte proportion de noeuds est faiblement connectée, alors qu'un très faible pourcentage de noeuds concentre à eux seuls un nombre élevé de connexions. D'une façon générale, la proportion $P(k)$ de noeuds dans le réseau ayant k liens peut être approché par :

$$c \times k^{-\lambda} \text{ avec } \lambda > 1$$

Plusieurs modèles ont été proposés pour la génération de réseaux scale-free. Le plus connu d'entre

eux est le modèle Barabasi-Albert [Barabasi 1999] qui introduit la notion d'attachement préférentiel.

(1) Un réseau initial contenant N_0 noeuds est tout d'abord créé. Dans ce réseau, nous devons garantir que $N_0 > 2$ et que $\forall v_i \in V, k_{v_i} > 1$; autrement dit, chaque noeud du réseau initial doit posséder au moins une connexion (le nombre initial de liens n'influence pas les propriétés résultantes).

(2) Une fois ce réseau initial obtenu, les actions suivantes sont itérées : un nouveau noeud v_i est ajouté au réseau et connecté à m autres noeuds v_j avec une probabilité p_j qui croît proportionnellement avec le degré k_{v_j} de v_j :

$$p_j = \frac{k_{v_j}}{\sum_{v_m \in V} k_{v_m}} \quad (2)$$

Naturellement, aux premières itérations, le nombre de liens est faible et la probabilité d'établir une connexion est plus ou moins équivalente pour tous les noeuds. Cependant, au fur et à mesure de l'évolution du réseau, on observe l'apparition de noeuds fortement connectés, avec lesquels un nouvel arrivant a une forte probabilité d'établir une connexion. Le mécanisme caractérisé par l'équation précédente est appelé attachement préférentiel. Il conduit à ce que les noeuds "préfèrent" établir une connexion avec les noeuds déjà fortement connectés.

Le réseau scale-free est aujourd'hui le type de réseau le plus fréquemment retrouvé dans les études menées sur les réseaux du monde réel. Un comparatif de ces différentes structures et de certaines mesures associées, inspiré de [27]⁶, est présenté sur la Figure 2.4.

6. Network science - Börner - 2007 -Annual Review of Information Science and Technology

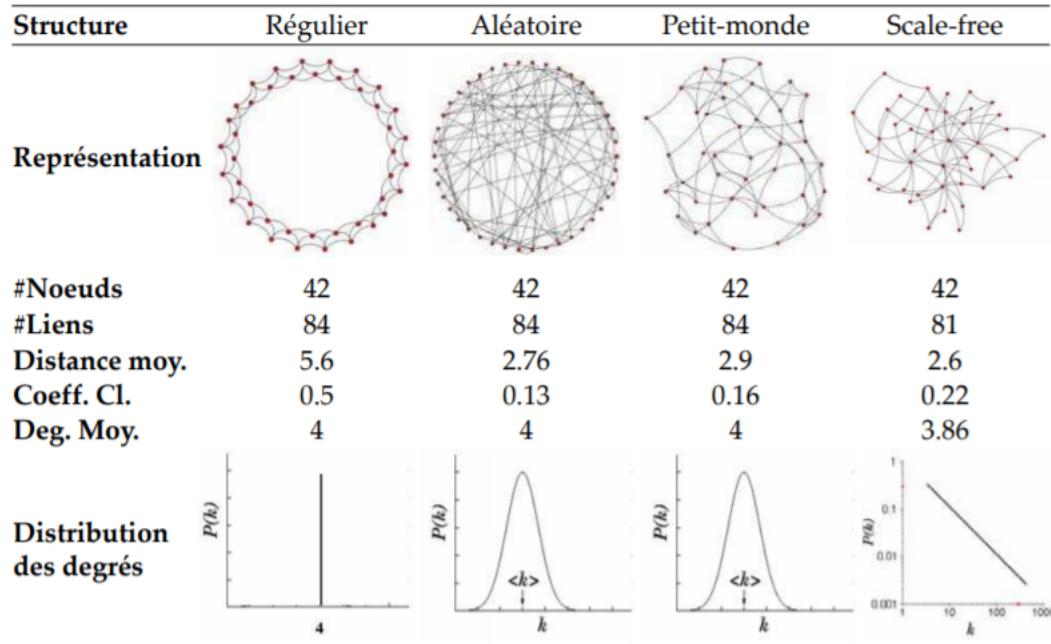


FIGURE 2.1-4 – Comparatif des différentes structures et mesures associées Source : Borner et al. [27]

2.2 Communautés et graphes

L'objectif de la détection de communautés dans les graphes, ou encore dans les réseaux sociaux, est de créer une partition des sommets, en tenant compte des relations qui existent entre les sommets dans le graphe, de telle sorte que les communautés soient composées de sommets fortement connectés et qu'elles soient peu reliées entre elles (Brandes et al., 2006⁷ ; Newman, 2004⁸ ; Schaeffer, 2007⁹ ; Lancichinetti et Fortunato, 2009¹⁰). Parmi les principales méthodes de détection de communautés proposées dans la littérature, on peut citer celles qui optimisent une fonction de qualité pour évaluer la qualité d'une partition donnée, comme la modularité, la coupe ratio, la coupe min-max ou la coupe normalisée (Kernighan et Lin, 1970¹¹ ; Chan et al., 1994¹² ; Shi et Malik, 2000¹³ ; Ding et al., 2001¹⁴ ; Newman et Girvan, 2004), les techniques hiérarchiques comme les algorithmes de division (Flake et al., 2000¹⁵), les méthodes spectrales (Von Luxburg, 2007¹⁶) ou l'algorithme de Markov et ses

7. Brandes, U.D. Delling, M. Gaertler, R. Goerke, M. Hoefer, Z. Nikoloski, and D. Wagner (2006), "Maximizing Modularity is hard,"ArXiv Physics e-prints.

8. J. Newman ,Finding community structure in very large networks

9. S.E. Schaeffer, Algorithms for nonuniform networks, Ph.D. Thesis, Helsinki University of Technology,

10. Andrea Lancichinetti, Santo Fortunato ,Community detection algorithms : a comparative analysis

11. Lin, Shen (1970). "An efficient heuristic procedure for partitioning graphs"

12. Obesity, Fat Distribution, and Weight Gain as Risk Factors for Clinical Diabetes in Men

13. Normalized Cuts and Image Segmentation Jianbo Shi and Jitendra Malik

14. Analyzing Information Flow in Brain Networks with Nonparametric Granger Causality

15. Community detection in Social MediaPerformance and application considerations

16. A tutorial on spectral clustering

extensions (Satuluri et Parthasarathy, 2009¹⁷). Ces techniques de partitionnement de graphes sont très utiles pour détecter des composantes fortement connectées dans un graphe

2.2.1 Définition de la communauté

Un réseau social peut souvent être décomposé en communautés, groupes d'entités communiquant beaucoup entre elles. Plusieurs définitions formelles de cette notion ont été proposées, mais l'intuition reste la même : il s'agit de décomposer le réseau social en sous-groupes tels que :

- (1) à l'intérieur d'un sous-groupe les noeuds soient très inter-connectés,
- (2) il existe peu de liens entre deux sous-groupes.

Un sous-groupe devrait donc être proche d'une clique (c'est-à-dire un ensemble de noeuds totalement connectés). La décomposition du réseau en communautés réalise donc une segmentation (clustering non supervisé en apprentissage statistique) des sommets. Alors que les techniques classiques de segmentation utilisent des mesures de similarité entre les noeuds, calculées à partir de leurs attributs, la détection de communautés est basée sur la structure du graphe sans prendre en compte les attributs (des méthodes hybrides considérant à la fois les attributs et les liens ont toutefois été proposées).

En 2002, Girvan et Newman (2002b) ont montré que la présence au sein de graphes sociaux de groupes de noeuds fortement connectés entre eux et faiblement avec le reste du graphe est une caractéristique des réseaux complexes, ils donnent le nom de communautés à ces groupes de noeuds fortement connectés. La difficulté de trouver des communautés est qu'elles peuvent avoir des ordres et des tailles différents au sein d'un même graphe. C'est en ce sens que nous allons dans la prochaine section formuler le problème général de détection de communautés et voir les différentes définitions des communautés (il n'existe pas de définition exacte).

2.2.2 Modularité

Newman a défini la modularité Q d'un graphe comme suit :

$$Q = \frac{1}{2m} \times \sum_{j,k} (a_{v_j,v_k} - \frac{d(v_j)d(v_k)}{2m}) \times \delta(v_j, v_k) \quad j = 1 \dots n \text{ et } k = 1 \dots n \quad (3)$$

Où m est le nombre de liens du graphe, n est le nombre de noeuds , a_{v_j,v_k} vaut 1 si les noeuds v_j et v_k sont liés et 0 dans le cas contraire. La variable $d(v_j)$ est le nombre de voisins du noeud v_j et δ est le symbole de Kronecker qui vaut 1 si v_j et v_k appartiennent à la même communauté et 0 sinon.

La modularité représente la différence entre la valeur d'adjacence entre deux noeuds de la même communauté (a_{v_j,v_k}) et la probabilité pour que ceux-ci soient connectés. Comme $\frac{d(v_j)d(v_k)}{2m}$ représente la probabilité qu'il existe un lien entre v_j et v_k , cette valeur est importante si le nombre de liens à

17. détection de communautés dans les réseaux sociaux utilisant liens et attribues

l'intérieur des communautés est élevé et le nombre de liens entre ces communautés est faible. Avec cette expression de modularité, une communauté est vue comme un ensemble de noeuds qui ont plus de liens entre eux que de liens avec des noeuds à l'extérieur de la communauté.

2.2.3 clustering

L'algorithme de clustering hiérarchique

Nous cherchons à regrouper les sommets similaires selon la distance r en communautés. Nous utilisons une approche de clustering hiérarchique agglomérative dans laquelle nous partons d'une partition initiale $P^0 = \{\{v\}, v \in V\}$ contenant n communautés composées d'un seul sommet. Nous réalisons alors n étapes successives de fusion de communautés qui créent une structure hiérarchique arborescente (aussi appelée dendrogramme). Le choix des communautés à fusionner se base sur les distances r_{ij} entre sommets que nous avons dénies précédemment.

Principe général

Nous considérons une partition initiale en communauté $P^0 = \{\{v\}, v \in V\}$ formée des n communautés composées d'un seul sommet. Nous allons générer une suite de partitions $(P^k)_{0 \leq k \leq n}$ en fusionnant successivement des communautés deux à deux. Le choix des communautés à fusionner repose sur les distances r entre les communautés adjacentes de la partition courante. Nous effectuons donc un calcul initial des distances r_{ij} entre chaque paire de sommets adjacents, ceci correspond donc à m distances calculées. Un processus itératif est alors répété jusqu'à l'obtention d'une partition finale $P^n = \{V\}$ ne possédant qu'une seule communauté regroupant tous les sommets du graphe. Chaque étape consiste à faire évoluer la partition courante P^k vers la partition P^{k+1} selon la démarche :

- Choisir deux communautés C_1 et C_2 de P^k . Ce choix est détaillé au paragraphe suivant.
- Fusionner ces deux communautés en une nouvelle communauté $C_3 = C_1 \cup C_2$ et créer la nouvelle partition $P^{k+1} = (P^k \setminus \{C_1, C_2\}) \cup \{C_3\}$.
- Mettre à jour les distances r qui ont changé, c'est à dire les distances $r_{C_3 C}$ pour toutes les communautés C adjacentes de la nouvelle communauté C_3 .

Chaque étape fusionne deux communautés pour en créer une nouvelle. Ceci construit une hiérarchie entre communautés qui peut être représentée par un arbre binaire. La racine de l'arbre correspond à la plus grande communauté V et les feuilles de l'arbre correspondent à tous les sommets $v \in V$. Chaque fusion crée un noeud de l'arbre avec deux sous-communautés filles incluses dans une plus grosse communauté. Nous donnons une représentation graphique de cette structure (aussi appelée dendrogramme). L'approche classique pour déterminer la partition en communautés sortie par l'algorithme consiste à choisir la partition P^k parmi les $n+1$ partitions $(P^k)_{0 \leq k \leq n}$ générées qui maximise la fonction de qualité choisie. Nous proposerons une meilleure approche permettant d'obtenir des partitions de meilleure qualité à partir du dendrogramme. Choix des communautés à fusionner Le point clé de l'algorithme est la manière de choisir les communautés à fusionner. Le nombre de fusions envisageables à chaque étape correspond au nombre de paires de communautés dans la partition courante P^k . Pour limiter le nombre de cas à considérer, nous n'allons considérer que des fusions entre communautés adjacentes (c'est à dire des commu-

nautés liées par au moins une arête). Cette heuristique raisonnable limite le nombre possible de fusions à m pour chaque étape. Ceci évite avantageusement de considérer de nombreux cas, il est en effet fortement improbable que deux communautés non reliées génèrent des marches aléatoires similaires.

2.3 Les algorithmes de détection de communautés

Dans cette section, nous présentons les algorithmes de détection de communautés que nous utilisons dans notre projet

2.3.1 Les approches de détection

La détection de communautés permet d'une part de comprendre les structures et les fonctionnements macroscopiques des grands graphes de terrain. D'autre part une telle méthode peut être utilisée comme brique de base pour l'élaboration d'algorithmes plus complexes comme la parallélisation, la visualisation ou la compression

Mise en contexte

Nous allons lister ici les principales approches qui ont été proposées à ce jour. Bien que la liste soit importante, elle est non exhaustive : afin d'en limiter la longueur, nous n'avons retenu que les approches qui ont reçu le plus d'attention de la part de la communauté scientifique. Notre but est de donner une vue d'ensemble des méthodes proposées, et d'en illustrer la diversité. Pour organiser la présentation, nous avons essayé de les regrouper en fonction du type d'approche utilisée. Tout d'abord les méthodes itératives (séparatives ou agglomératives) qui divisent ou fusionnent successivement des communautés représentent les approches les plus répandues. Nous allons aussi nous intéresser aux méthodes utilisant des marches aléatoires, qui jouent un rôle important dans ce projet car nous allons nous-même fonder notre approche sur les marches aléatoires. Nous finirons par présenter diverses autres approches qui ne rentrent pas dans les deux premières catégories.

Les approches classiques

La détection de communautés s'approche des deux thématiques classiques en informatique que sont le partitionnement de graphe et le clustering de données. La première, initialement introduite pour la parallélisation de processus, cherche à répartir des tâches, représentées par les sommets d'un graphe, tout en minimisant les échanges, représentés par les arêtes. La seconde thématique de clustering de données est une thématique générale plus vaste dans laquelle on cherche à regrouper des données possédant des caractères communs. Cette problématique possède des applications dans de très nombreux domaines et on peut considérer la détection de communautés comme une de ses applications pour les grands graphes de terrain.

Le partitionnement de graphe

Le but du partitionnement de graphe est de grouper les sommets d'un graphe en un nombre prédéterminé de parties (et de tailles elles aussi prédéterminées) tout en minimisant le nombre d'arêtes tombant entre les différents groupes. Cette approche ne convient pas totalement à la détection de communautés car elle a l'inconvénient de requérir une connaissance préalable du nombre de communautés recherchées ainsi que de leurs tailles. Nous citons

les deux méthodes générales ayant eu le plus de succès.

La méthode de bisection spectrale

Elle consiste à calculer le vecteur propre correspondant à la plus petite valeur propre non nulle de la matrice Laplacienne du graphe, $L = D - A$ ¹⁸. Le graphe est alors séparé en deux parties en fonction du signe de leur composante selon ce vecteur propre. Cette méthode fonctionne en temps $O(n^3)$ et obtient de bons résultats lorsque le graphe possède effectivement deux grandes communautés de tailles similaires, ce qui n'est qu'un cas particulier dans l'optique de détection de communautés.

La méthode de Kernighan et Lin

Il s'agit d'un algorithme de bisection visant à trouver la coupe du graphe minimisant le nombre d'arêtes tombant entre les deux groupes. L'algorithme nécessite comme paramètre la taille des communautés à détecter. Une coupe de la bonne taille est choisie arbitrairement comme point de départ en utilisant une heuristique gloutonne. La bisection est améliorée itérativement en faisant des échanges de sommets entre les communautés. A chaque étape on échange les deux sommets procurant la meilleure réduction du nombre d'arêtes externes, avec la condition imposée de ne jamais changer deux fois un même sommet. Il y a donc exactement $\frac{n}{2}$ étapes de calcul et la meilleure partition rencontrée au cours du déroulement de l'algorithme est retenue. La complexité du pire cas est en $O(n^3)$.

Le clustering de données

Cette problématique a été introduite pour analyser des données. Elle cherche à partitionner des données en se basant sur une mesure de distance (ou de similarité) entre celles-ci. Cette distance représente la similarité des données et on cherche à former des groupes d'objets proches les uns des autres. Nous renvoyons le lecteur à l'article de synthèse [43] comme point d'entrée dans ce très vaste sujet. Cette problématique peut être modélisée par des graphes pondérés : chaque donnée est un sommet et les arêtes sont pondérées par la distance entre ces données. Le problème de détection de communautés peut être vu comme un problème de clustering de données pour lequel il faut choisir une distance adéquate. Cependant, les graphes considérés par les applications usuelles de clustering ne possèdent pas les caractéristiques spécifiques des grands graphes de terrain. Par conséquent de nombreuses approches classiques de clustering de données sont inadaptées aux grands graphes de terrain, principalement pour des questions de taille. En pratique seules les méthodes de clustering hiérarchique ont permis d'obtenir de bons résultats. Nous allons donc présenter ces méthodes, qui ont été adaptées pour la détection de communautés par plusieurs approches récentes.

Les méthodes de clustering hiérarchique.

L'algorithme part d'une structure dans laquelle chaque sommet est identifié comme une petite communauté. On itère alors les étapes suivantes : on calcule des distances entre communautés et on fusionne les deux communautés les plus proches en une nouvelle communauté. Le nombre de communautés est réduit de un à chaque étape, et le processus s'arrête lorsqu'il n'y a plus qu'une seule communauté correspondant au graphe entier. On obtient ainsi une structure hiérarchique de communautés qui peut être représentée sous une forme arborescente appelée

18. Laurent Beauguitte, Pierre Beauguitte ; Opérations matricielles et analyse de graphe

dendrogramme : les feuilles sont les sommets du graphe tandis que les noeuds représentent les communautés créées et sont reliés en fonction des fusions de communautés. La racine de la structure correspond au graphe entier. Il existe plusieurs façons de définir la distance entre deux communautés. La plus simple (single linkage) considère que la distance entre deux communautés est la distance minimale entre deux sommets de celles-ci. A l'opposé, on peut considérer la distance maximale (complete linkage). De manière intermédiaire (average linkage) on peut considérer que la distance entre deux communautés est la moyenne des distances entre chaque paire de leurs sommets. Il est encore possible dans certains cas de représenter chaque communauté par un sommet moyen et de considérer leurs distances respectives (centroïd). La dernière méthode (dite de Ward), consiste à minimiser la somme des carrés des distances de chaque sommet au sommet moyen représentant sa communauté.

Les approches séparatives

L'idée commune à toutes ces méthodes est d'essayer de scinder le graphe en plusieurs communautés en retirant progressivement les arêtes reliant des communautés distinctes. Les arêtes sont retirées une à une, et à chaque étape les composantes connexes du graphe obtenu sont identifiées à des communautés. Le processus est répété jusqu'au retrait de toutes les arêtes. On obtient alors une structure hiérarchique de communautés (dendrogramme), comme pour les méthodes de clustering hiérarchique. Les méthodes existantes diffèrent par la façon de choisir les arêtes à retirer.

Les approches de Radicchi et al [68] et d'Auber et al [6] basées sur le clustering d'arêtes

La détection des arêtes intercommunautaires est ici basée sur le fait que de telles arêtes sont dans des zones peu clusterisées. Radicchi et al [68] proposent un coefficient de clustering (d'ordre g) d'arêtes. Il est défini comme étant le nombre de cycles de longueur g passant par l'arête divisé par le nombre total de tels cycles possibles (étant donné les degrés des extrémités de l'arête). Cet algorithme retire donc à chaque étape l'arête de plus faible clustering (d'un ordre donné, 3 ou 4 en pratique). Chaque suppression d'arête ne demande alors qu'une mise à jour locale des coefficients de clustering, ce qui lui permet d'être bien plus rapide que le précédent algorithme. La complexité totale est en $O(m^2)$. L'approche d'Auber et al [6] utilise un clustering d'arêtes d'ordre 3 pour proposer un outils de visualisation de graphes. L'algorithme de Fortunato et al basé sur la centralité d'information [29]. Il s'agit d'une variante de l'algorithme de Girvan et Newman basé sur une autre notion de centralité : la centralité d'information. Les auteurs définissent l'efficacité de communication C_{ij} entre deux sommets i et j du graphe comme étant l'inverse leur distance (en terme de plus courts chemins). L'efficacité E du réseau est alors définie comme la moyenne des C_{ij} pour tous les couples de sommets. Enfin, la centralité d'information d'une arête est définie comme étant la diminution relative de l'efficacité du réseau lorsque l'on retire cette arête du graphe. Cette approche donne des résultats à une complexité en $O(m^3 \times n)$.

Les approches agglomératives

L'idée commune de toutes ces méthodes est d'utiliser une approche, s'apparentant à celle du clustering hiérarchique, dans laquelle les sommets sont regroupés itérativement en communautés en partant d'une partition de n communautés composées d'un seul sommet. Les regroupements de communautés sont poursuivis jusqu'à obtenir une seule communauté regroupant tous les sommets, et une structure hiérarchique de communautés (dendrogramme) est ainsi construite. Ces approches sont similaires aux approches séparatives à la différence qu'elles travaillent de bas en haut dans la hiérarchie des communautés au lieu de travailler de haut en bas

L'algorithme d'optimisation de la modularité proposé par Newman [58] et amélioré par Clauset et al [16].

Newman introduit une notion de modularité; il s'agit d'une fonction Q mesurant la qualité d'une partition du graphe en communautés. Nous présentons cette fonction dans notre implémentations (elle se base sur les proportions d'arêtes internes aux communautés et les proportions d'arêtes liées à chaque communauté). Affin de maximiser cette quantité l'algorithme glouton proposé fusionne à chaque étape les communautés permettant d'avoir la plus grande augmentation de la modularité. Pour améliorer les performances de l'algorithme, seules les communautés ayant une arête entre elles peuvent être fusionnées à chaque étape. Chaque fusion se fait en $O(n)$ et la mise à jour des valeurs des variations de Q (pour chaque nouvelle fusion possible) peut être effectuée facilement en $O(m)$. La complexité globale est donc $O((m + n)n) = O(mn)$. Cette méthode est très rapide et permet de traiter de très grands graphes. La qualité des partitions obtenues est cependant moins bonne qu'avec des méthodes plus coûteuses. La complexité de cette méthode a été améliorée par Clauset [16] en utilisant une structure de données adaptée. Une autre optimisation de cette méthode a récemment été proposée par Wakita et Tsurumi [80] an de traiter des graphes de taille encore plus importante.

L'algorithme de Donetti et Muñoz basé sur les propriétés spectrales de la matrice Laplacienne du graphe [19, 20]

Les propriétés spectrales de la matrice Laplacienne du graphe, $L = DA$ sont utilisées pour détecter des communautés. En effet les coordonnées i et j des vecteurs propres correspondant aux plus petites valeurs propres non nulles sont corrélées lorsque les sommets i et j sont dans la même communauté. Une distance entre sommets est alors calculée à partir de ces vecteurs propres, cette distance étant ensuite utilisée dans un algorithme de clustering hiérarchique. Le nombre de vecteurs propres à considérer est a priori inconnu. Plusieurs calculs sont successivement effectués en prenant en compte différents nombres de vecteurs propres, et le meilleur résultat est retenu. Les performances de l'algorithme sont limitées par les calculs des vecteurs propres qui se fait en $O(n^3)$ pour une matrice creuse. Une amélioration de cette approche a été proposée et utilise une version normalisée de la matrice Laplacienne [20]. Notons que les vecteurs propres de cette nouvelle matrice sont les mêmes vecteurs propres que ceux de la matrice de transition des marches aléatoires

Les approches utilisant des marches aléatoires

Les marches aléatoires dans les graphes sont des processus aléatoires dans lesquels un marcheur est positionné sur un sommet du graphe et peut à chaque étape se déplacer vers un des sommets voisins. Le comportement des marches aléatoires est étroitement lié à la structure du graphe, ainsi plusieurs approches de détection de communautés se basent sur ces comportements. Nous avons souhaité distinguer les approches utilisant les marches aléatoires pour les mettre en parallèle de l'approche que nous proposerons qui est elle aussi basée sur des marches aléatoires. Notons que beaucoup des algorithmes présentés dans cette section sont aussi des algorithmes agglomératifs qui auraient pu être classés dans la section précédente.

Markov Cluster Algorithm de van Dongen [79].

Cette approche calcule des probabilités de transition entre tous les sommets du graphe en partant de la matrice de transition des marches aléatoires. Deux opérations matricielles sont successivement itérées. La première calcule les probabilités de transition par des marches aléatoires de longueur fixée r et correspond à une élévation de la matrice à la puissance r . La seconde consiste à amplifier les différences en augmentant les transitions les plus probables et en diminuant les transitions les moins probables. Les transitions entre sommets d'une même communauté sont alors favorisées et les itérations successives des deux opérations conduisent à une situation limite dans laquelle seules les transitions entre sommets d'une même communauté sont possibles. La complexité totale de l'algorithme est en $O(n^3)$

approches basées sur le temps moyen pour atteindre un sommet [88, 86].

Pour mesurer la proximité structurelle de deux sommets, ces approches utilisent le nombre moyen d'étapes pour qu'une marche aléatoire atteigne un sommet donné en partant d'un autre sommet. Cette longueur moyenne peut être calculée grâce à une inversion de matrice en $O(n^3)$. Les deux approches, l'une proposée par Zhou et Lipowsky [88] et l'autre proposée par Yen et al [86] utilisent cette quantité pour définir deux distances mesurant la similarité des sommets. Cette distance est utilisée dans la première approche dans un méthode de clustering hiérarchique pour constituer un algorithme de détection de communautés nommé netwalk. La seconde approche utilise leur distance dans un algorithme de clustering k-mean.

Il existe de nombreuses autres méthodes reliées à la problématique de détection de communautés dans les grands graphes de terrain. Sans en établir une liste exhaustive

2.3.2 Les Algorithmes étudiés

Le nombre d'algorithmes comparés est une limite indéniable de notre étude, néanmoins il est impossible d'évaluer toutes les méthodes publiées. Nous avons choisi des méthodes récentes et reconnues comme efficaces. Afin de les tester sur de grands graphes, il était nécessaire que ces algorithmes soient performants en terme de temps de calcul et de consommation mémoire (par exemple, l'algorithme historique de Girvan et Newman [02], ayant une complexité en $O(n^3)$, ne peut-être utilisé sur des graphes pouvant contenir des milliers de sommets). Enfin, nous avons choisi des algorithmes représentants différentes approches. Deux d'entre eux sont basés sur des marches aléatoires (WalkTrap et InfoMap), deux sur une optimisation de la modularité (Louvain et FastGreedy) et un dernier sur une recherche de motifs locaux (CFinder). L'objectif étant de faire éventuellement apparaître des similitudes ou différences entre ces approches. Pour un état de l'art plus complet des méthodes existante nous renvoyons à [SCH 07, FOR 10].

CFinder [12] : Cette méthode est basée sur une recherche de motifs locaux. Une communauté est définie comme une chaîne de k -cliques adjacentes. Une k -clique est un sous-ensemble de k sommets tous adjacents les uns aux autres, et deux k -cliques sont adjacentes si elles partagent $k - 1$ sommets. L'avantage immédiat d'une telle approche est la détection de communautés avec recouvrement, un sommet pouvant appartenir à plusieurs k -cliques non forcément adjacentes. C'est la seule méthode évaluée ici qui permette un recouvrement des communautés. Une limite de cet algorithme est qu'il nécessite un paramétrage : la valeur de k (la taille des communautés à considérer). Nous avons choisi une valeur de $k = 4$, généralement acceptée comme la plus efficace. Les tests que

nous avons menés semblent montrer que modifier cette valeur influe fortement sur les résultats trouvés mais peu sur la tendance générale de ces résultats comparés aux autres algorithmes.

Walktrap : [14] : Utilise une distance entre sommets basée sur des marches aléatoires [17]. Une marche aléatoire courte, partant d'un sommet donné, tend à rester dans la (les) communauté(s) de ce sommet. Ainsi la distance entre les résultats de deux marches aléatoires partant de deux sommets distincts, révèle efficacement l'appartenance commune ou non de ces sommets à une même communauté. Cette distance permet à cette méthode de partitionner le graphe par l'intermédiaire d'un algorithme de clustering hiérarchique

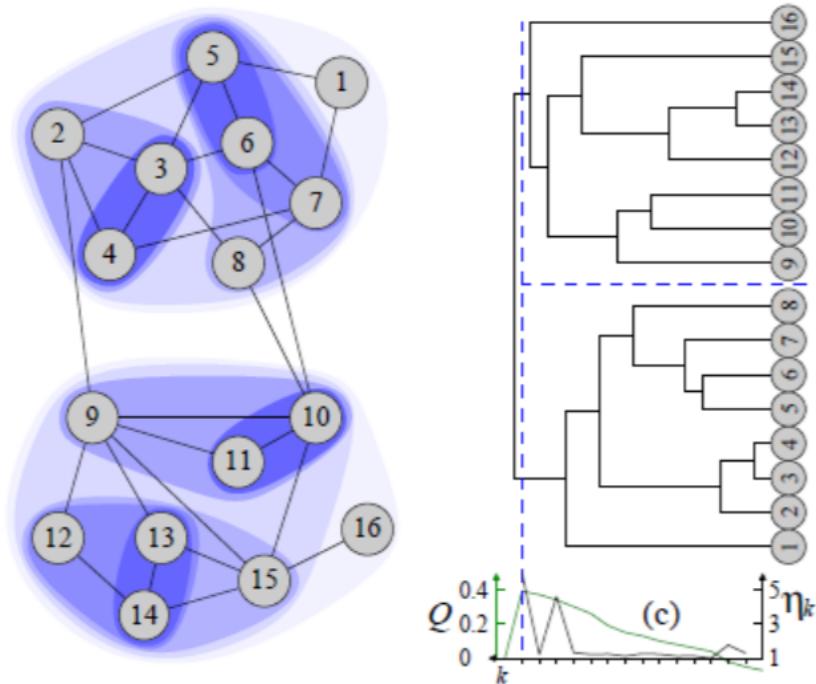


FIGURE 2.3-1 – Exemple de réseau à 16 sommets divisé en 2 communautés avec Walktrap

La complexité de **Walktrap** est de $O(nmH)$ où m est le nombre de liens, n le nombre de noeuds et H est la hauteur du dendrogramme. Dans le cas où les sommets sont fusionnés un par un, H atteint son maximum et vaut $n - 1$. L'inconvénient de cet algorithme cependant est le paramètre t qui représente le nombre de pas ou de liens entre les noeuds.

En effet, l'algorithme doit fixer à l'avance cette valeur. En pratique, on affecte la valeur $\log(n)$ à $t(t = \log(n))$. Toutefois cette valeur est liée à la densité des liens dans un graphe et non pas à sa taille

Louvain : [08] : Méthode d'optimisation de la modularité. La modularité est une fonction de mesure de la qualité d'un partitionnement, introduite initialement par Girvan et Newman, pour choisir une coupe privilégiée dans un dendrogramme issu d'un clustering hiérarchique [02]. La modularité mesure le nombre d'arêtes à l'intérieur des communautés moins ce même nombre obtenu sur un graphe aléatoire (c.a.d. sans structure) de même taille mais gardant exactement la même distribution de degrés. L'idée est que dans un graphe aléatoire, certains liens ont naturellement une forte chance d'exister (notamment entre 2 noeuds de très fort degrés). L'existence de ce lien dans le graphe réel ne sera donc pas un argument pertinent pour considérer que ces 2 noeuds sont effectivement dans la même communauté. Au contraire, l'existence d'un lien entre 2 noeuds ayant peu de chances d'être liés dans le graphe aléatoire sera un argument fort pour les regrouper. Louvain utilise une méthode d'optimisation gloutonne de la modularité. Au départ chaque sommet est dans sa propre communauté, puis chaque sommet prend la communauté d'un de ses voisins de tel sorte que le gain de modularité soit maximal. Cette opération est répété plusieurs fois sur l'ensemble des sommets jusqu'à ce qu'aucun gain ne soit possible. Toute l'opération est ensuite répétée sur le graphe des communautés.

Dans l'algorithme de Louvain, la première phase utilise l'heuristique VM (vertex mover). À partir d'un graphe d'origine à partitionner $G^0=G$, l'application de VM au premier niveau 0 donne une partition P^0 . Le graphe de niveau 1, noté G^1 est engendré à partir de P^0 selon la procédure décrite plus bas. Au niveau 1, l'heuristique VM est appliquée à G^1 et cette procédure récursive se poursuit jusqu'au dernier niveau L où VM ne déplace aucun noeud, c'est-à-dire lorsque $|P^L| = |G^L|$.

Pour passer d'un niveau l au niveau supérieur $l + 1$, chaque noeud de G^{l+1} est associé à une communauté de P^l par une fonction que nous notons $T^l : P^l \rightarrow G^{l+1}$. La construction de G^{l+1} en fonction de G^l et de P^l suit les règles suivantes :

1. À chaque communauté C de P^l est associé un noeud $T^l(C)$ dans G^{l+1}
2. Une arête existe entre les noeuds $T^l(C_1)$ et $T^l(C_2)$, représentant les communautés C_1 et C_2 de P^l , s'il existe au moins une arête ayant une extrémité dans C_1 et l'autre dans C_2 .
3. Le poids d'une arête entre les noeuds $T^l(C_1)$ et $T^l(C_2)$ est égal à la somme des poids des arêtes entre les deux communautés soit $d(C_1, C_2)$.
4. Une boucle est ajoutée à chaque noeud $T^l(C)$ avec un poids égal au degré interne de la communauté C dans P^l , soit $d_{in}(C)=d(C, C)$ soit encore $2 \sum_{u \in C \text{ et } v \in C} w(u, v)$

Cette procédure est illustrée par la figure 2.6 . Nous obtenons finalement une partition hiérarchique dans laquelle les communautés de niveau L contiennent des sous-communautés de niveau inférieur $L - 1$ et ainsi de suite jusqu'aux noeuds du graphe d'origine. Cet algorithme est le plus rapide pour optimiser la modularité avec une complexité constatée sur des instances de graphes peu denses en $O(m)$, qui autorise le traitement de très grands graphes. Son optimisation de Q est correcte, mais n'égale pas celle des méta-heuristiques et souffre surtout de dégénérescence comme nous le mettrons en évidence par la suite. Toutefois, cet algorithme fournit un compromis très intéressant entre performance et vitesse d'optimisation pour une utilisation dans les algorithmes d'optimisation poussée que nous avons développés.

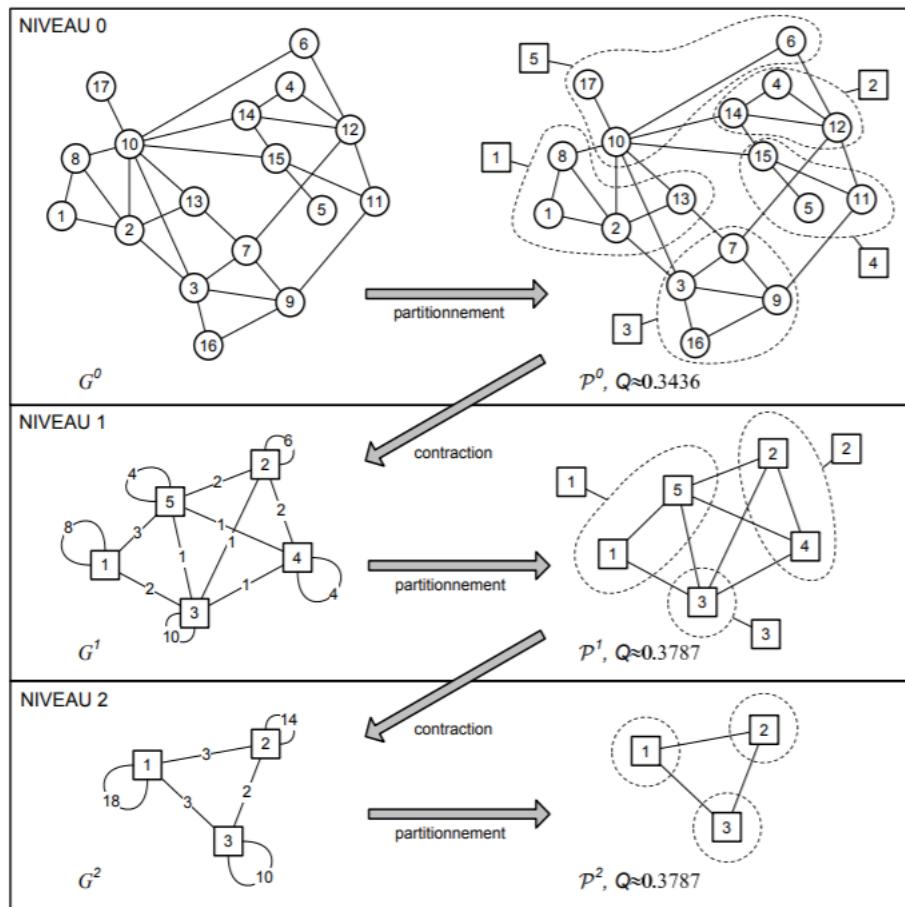


FIGURE 2.3-2 – Illustration de l'algorithme de Louvain

Explication du figure 2.3.2 :

Au niveau 0, le partitionnement par l'heuristique VM du graphe G^0 donne un découpage en cinq communautés (partition P^0). Dans le graphe contracté du niveau suivant, il y a cinq noeuds (un par communauté du niveau précédent) et les arêtes sont pondérées de façon à ce que la modularité soit la même, si l'on prend des communautés singlons (un noeud dans chacune) dans le nouveau graphe. Le processus se poursuit avec ce graphe jusqu'au niveau 2 où la modularité de la partition singleton de départ ne peut pas être améliorée.

Label Propagation : Cet algorithme décrit en [U. N. Raghavan, 2007] se base sur le principe que chaque noeud change de communauté en fonction de la communauté à laquelle appartiennent ses voisins. Un noeud fait partie de la communauté qui contient le plus grand nombre de noeuds voisins. Ce processus est le modèle d'apprentissage pour Label Propagation, qui s'exécute sur tous les noeuds dans chaque itération. Au début de l'algorithme, chaque noeud se trouve dans une seule communauté. Puis, les noeuds changent de communautés tout en respectant le modèle d'apprentissage. Avec cette méthode, un groupe de noeuds, fortement liés entre eux, finit par se trouver dans la même communauté. La communauté d'un noeud v_i à la t ème itération dépend des communautés des voisins de v_i à la $t - 1$ ème itération. Il existe aussi une autre façon asynchrone pour affecter les communautés. La communauté d'un noeud v_i à la t ème itération dépend des communautés des voisins de v_i à la t ème itération. Idéalement, l'algorithme s'arrête lorsque les noeuds ne changent plus de communautés même si ce n'est pas toujours le cas. En effet, lorsqu'un noeud a un nombre maximal de voisins qui appartiennent à deux communautés ou plus, il change de communauté à chaque itération. L'algorithme Label Propagation est décrit principalement par les étapes suivantes :

- 1) Initialement chaque noeud v_i , à l'itération $t = 0$, appartient à une seule communauté $i : C_{v_i}(0) = i$, $i = 1 \dots n$
 - 2) Définir $t = 1$
 - 3) Pour chaque noeud v_i , trouver la communauté du noeud v_i à la t ème itération $C_{v_i}(t)$ en fonction des communautés de voisins de v_i à l'itération t ou $t-1$.
 - 4) Si tous les noeuds ne changent plus de communautés $C_{v_i}(t) = C_{v_i}(t - 1)$ pour $i = 1 \dots n$, l'algorithme s'arrête. Sinon, $t = t + 1$ et aller à l'étape (3).
- Comme l'algorithme choisit le noeud à traiter aléatoirement et avec une condition d'arrêt (non pas une mesure), plusieurs résultats finaux peuvent être obtenus.

Fast Greedy : L'algorithme utilise la fonction de modularité Q définie au début de cette section. Chaque division possible du graphe a une valeur propre Q . Cette valeur est élevée pour une bonne division du graphe est faible dans le cas contraire. En observant toutes les divisions possibles du graphe, on remarque que le calcul exhaustif de Q prend une quantité du temps exponentiel au nombre de noeuds. Cette méthode peut être utilisée pour les réseaux de taille faible (une vingtaine ou trentaine de noeuds) pour avoir un temps d'exécution raisonnable. Des stratégies d'optimisation ont été proposées afin d'améliorer les délais d'exécution de cet algorithme. La présente approche commence par considérer chaque noeud comme une communauté à part, pour ensuite fusionner les communautés en paires, en choisissant à chaque étape la paire de communautés dont la fusion donne la plus forte augmentation de Q par rapport à l'itération antérieure. Puisque la fusion des communautés non adjacentes ne produit jamais une plus grande augmentation de Q , plusieurs fusions possibles sont évitées. Dans ce cas, il y a m paires et $n - 1$ fusions possibles pour relier toutes les communautés entre-elles. Ainsi, le temps total d'exécution est en $O(mn)$. La valeur maximale de Q correspond à la meilleure structure de communautés. Fast Greedy n'est

pas un algorithme hiérarchique. Il n'accorde aucun score ou paramètre aux liens. Ajoutons à cela qu'il ne mesure pas la distance ou la similarité entre les communautés à fusionner. De même, l'inconvénient majeur de cette méthode réside dans sa qualité de classification qui est moins bonne en comparaison avec d'autres algorithmes utilisant la modularité, citons par exemple Edge Betweenness.

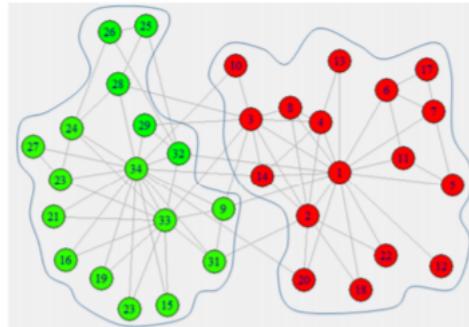


FIGURE 2.3-3 – l'exécution de l'algorithme fast greedy sur Le réseau social du club de karaté de Zachary

3 Graphes aléatoires

Nous présenterons, Dans ce chapitre, le but de l'utilisation des graphes aléatoires. D'abord, dans la section 3.2, nous rappellerons les notions mathématiques nécessaires et leur intérêt dans le monde des graphes aléatoires GA . Ensuite, nous décrirons les caractéristiques des graphes aléatoires dans la section 3.3. Enfin, nous définirons dans la section 3.4, le modèle d'Erdos-Rényi qui est Le premier modèle des graphes aléatoires. Et pour finir, nous terminerons ce chapitre par quelques opérations sur GA.

3.0.1 Définition du graphe aléatoire

Un graphe aléatoire est un graphe qui est généré par un processus aléatoire. Les graphes aléatoires ont été introduits par Paul Erdős et Alfréd Rényi en 1959 afin de prouver certains résultats sur les graphes. Plusieurs modèles de graphes théoriques existent. Le choix du modèle doit être fait avec soin car les propriétés des graphes peuvent être très différentes selon le modèle choisi . Un graphe aléatoire de taille n est un graphe à n sommets dont nous avons choisi aléatoirement les arrêtes. Le modèle d'Erdős-Rényi consiste à considérer que l'existence de chaque arc est indépendante de celle des autres et que chaque arc a une probabilité p d'exister . Dans ce cas, la distribution des degrés des sommets du graphe suit une loi de Poisson (une loi binomiale). Ces graphes sont généralement notés G(n,p) où n est le nombre de sommets et p la probabilité que les arêtes soient présentes. Néanmoins, peu de graphes dans la nature suivent cette distribution . De plus, le modèle Erdős-Rényi crée des graphes presque sûrement connexes alors qu'ils sont censés être aléatoires.

3.0.2 Pourquoi les graphes aléatoires ?

pour prédire les tendances futures des données. Une telle analyse peut aider à nous fournir une meilleure compréhension des données en général. La classification prédit les étiquettes catégorielles (discrètes, non ordonnées), le modèle de prédiction et la fonction de valeur continue.

3.1 Rappels mathématiques

3.1.1 Probabilité et ses lois

Dans cette partie on va traiter toutes les notions dont on aura besoin et qui sont en relation avec la probabilité dont lesquelles Paul Erdős et Alfréd Rényi introduisent leur premier modèle des graphes aléatoires

Définition : Une probabilité (ou mesure de probabilité) P sur Ω est une application sur l'ensemble des événements telle que

- (1)- $P(\Omega) = 1$
- (2)- pour tout événement A , $0 \leq P(A) \leq 1$
- (3)-pour toute suite $A_1, A_2\dots$ (i.e pour tout $i \neq j, A_i \cap A_j = \emptyset$)

$$P(\bigcup_{i=0}^{\infty} A_i) = \sum_{i=0}^{\infty} P(A_i)$$

Une probabilité est une mesure dans le sens où plus un événement est grand plus sa probabilité est importante.

Variables aléatoires discrètes :

On dit qu'une variable aléatoire est discrète si elle ne prend qu'un nombre fini ou dénombrable de valeurs :

$$X \in \{x_k, k \in K \subset \mathbf{N}\}$$

Dans ce cas, la loi de la variable aléatoire X est la loi de probabilité sur l'ensemble des valeurs possibles de X qui affecte la probabilité $P[X = x_k]$ au singleton $\{x_k\}$.

En pratique, l'ensemble des valeurs que peut prendre X est N ou une partie de N .
Déterminer la loi d'une variable aléatoire discrète c'est

- 1)-Déterminer l'ensemble des valeurs que peut prendre X .
- 2)-Calculer $P[X = x_k]$ pour chacune de ces valeurs x_k .

Les lois discrètes qui sont utilisées dans les graphes aléatoires.

Loi uniforme. La loi uniforme sur un ensemble fini est la loi des "tirages au hasard" dans cet ensemble, ou équiprobabilité. Elle donne la même probabilité $1/n$ à tous les éléments de l'ensemble, s'il est de cardinal n .

Loi de Bernoulli. Les plus simples des variables aléatoires discrètes sont les indicatrices d'événements. Si A est un événement de probabilité p , la variable aléatoire 1_A prend la valeur 1 si A est réalisé, et 0 sinon. Sa loi est la loi de Bernoulli de paramètre p .

$$P[1_A = 0] = 1 - p \quad , \quad P[1_A = 1] = p .$$

Les deux autres exemples de base sont la loi binomiale et la loi géométrique.

Loi binomiale. On répète la même expérience n fois indépendamment et on compte le nombre de fois où l'événement A se produit. On considérera la répétition des n expériences comme une nouvelle expérience globale. Comme seul l'événement A nous importe, on pourra ne retenir de l'expérience globale qu'un n -uplet de booléens du type :

$$(A, \bar{A}, A, A, \bar{A}, \dots, \bar{A}, A),$$

qu'il sera plus simple de transformer en un n -uplet de 0 et de 1. Notons :

- $X_i = 1$ si A est vrai à l'issue de la i -ème expérience, $X_i = 0$ sinon.
- $S_n = \sum_{i=1}^n X_i$ le nombre de fois où A est réalisé au cours des n expériences.

Notons p la probabilité de l'événement A . La variable aléatoire X_i suit la loi de Bernoulli de paramètre p . La variable aléatoire S_n prend ses valeurs dans l'ensemble $\{0, \dots, n\}$. Pour déterminer sa loi, ce sont les événements du type " $S_n = k$ " qui nous intéressent.

Du fait de l'hypothèse d'indépendance des expériences, la probabilité d'un résultat quelconque de l'expérience globale est un produit de probabilités. Par exemple :

$$P[(A, \bar{A}, A, A, \bar{A}, \dots, \bar{A}, A)] = p(1-p)p p(1-p) \dots (1-p)p .$$

Tout n -uplet particulier contenant k "1" (et $n-k$ "0") a pour probabilité $p^k(1-p)^{n-k}$. Il y en a :

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} ,$$

qui est le nombre de manières de choisir k indices parmi n . D'où :

$$P[S_n = k] = \binom{n}{k} p^k (1-p)^{n-k} , \quad \forall k = 0, \dots, n .$$

Alors On dit qu'une variable aléatoire X suit la loi binomiale de paramètres n et p (notée $\mathcal{B}(n, p)$) si :

1. X prend ses valeurs dans l'ensemble $\{0, 1, \dots, n\}$

$$2. P[X = k] = \binom{n}{k} p^k (1-p)^{n-k} , \quad \forall k = 0, \dots, n .$$

Loi de Poisson. De nombreuses variables aléatoires discrètes correspondent à des comptages d'objets possédant un caractère relativement rare dans un grand ensemble : atomes d'un isotope, molécules d'un élément chimique, bactéries, virus, individus porteurs d'un gène particulier,... On utilise souvent une loi de Poisson comme modèle pour ces comptages.

Une variable aléatoire suit la loi de Poisson de paramètre $\lambda > 0$ si elle prend ses valeurs dans N , et si pour tout $k \in N$,

$$P[X = k] = e^{-\lambda} \frac{\lambda^k}{k!} .$$

3.2 Modèle d'Erdős-Rényi

3.2.1 Définition générale

Le premier modèle des graphes aléatoires qui a été utilisé est celui de Erdős et Rényi en 1959 [1]. Il est l'un des modèles les plus simples qu'on puisse imaginer. On considère N sommets, et on relie chaque couple de sommets distincts par un arc avec une probabilité p indépendamment les uns aux autres.

On voit immédiatement qu'un tel modèle est trop simple pour pouvoir modéliser des réseaux de la vie réelle. Par exemple si on modélise la relation d'amitié par un graphe, on imagine facilement que les relations d'amitié ne sont pas indépendantes les une aux autres. Les réseaux de régulation génétique sont un autre exemple. Uri Alon [3] a montré l'existence des petites structures topologiques appelée motifs (par exemple des triangles de régulation) qui apparaissent plus fréquemment dans les réseaux biologiques que dans des réseaux aléatoires. A fortiori, on se rend compte que ce modèle est très pauvre pour modéliser des structures topologiques de cette espèce.

On fixe un entier $n \geq 1$, et un paramètre $p = p_n \in [0, 1]$. Le graphe aléatoire d'ErdősRényi $G = G(n, p)$ est le graphe dont les sommets sont les entiers $i \in [1, n]$, et tel que les variables aléatoires

$$X_{ij} = \begin{cases} 1 & \text{Si } \{i, j\} \text{ est une arête de } G \\ 0 & \text{Sinon} \end{cases}$$

avec $1 \leq i \leq j \leq n$ sont des variables de Bernoulli indépendantes de même paramètre p : $P[X_{ij} = 1] = 1 - P[X_{ij} = 0] = p$. Autrement dit, chaque arête possible entre les sommets de G apparaît avec probabilité p , indépendamment des autres arêtes.

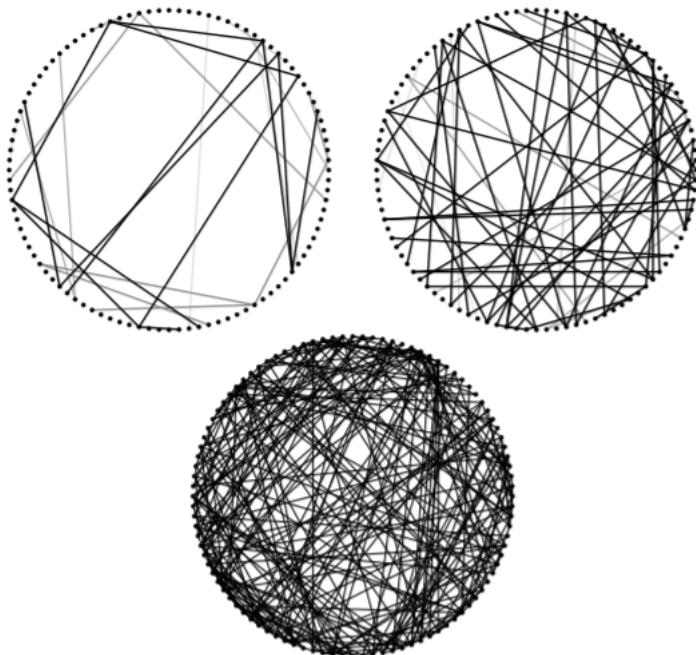


FIGURE 3.2-1 – Trois graphes aléatoires d’Erdős-Rényi

Explication du figure 3.2.1

les trois graphes sont de taille $n=100$ et paramètres $p=0.007, 0.015$ et 0.05 . Lorsque on fait grandir la probabilité, le nombre des arcs s’augmenter

3.2.2 L'avantage du modèle Erdős-Rényi

L'avantage du modèle de Erdős-Rényi est qu'il est relativement simple à comprendre et à étudier. Par exemple on peut facilement s'intéresser à la loi de la probabilité des degrés des sommets. Un sommet est de degré k s'il est absolument relié à k autres des $N-1$ sommets du graphe.

3.2.3 Implémentation du modèle Erdős-Rényi

Pour implémenter le code ci-dessous, il faudra installer la bibliothèque networkx et la bibliothèque matplotlib. Ensuite, Apple fonction prédéfinie de la bibliothèque networkx.

Erdos_renyi_graph(n,p,seed=None, directed=False)

Renvoie un graphe aléatoire $G(n,p)$, également appelé graphe d’Erdos-Rényi ou binomial.

Le modèle $G(n,p)$ choisit chacune des arêtes possibles avec une probabilité p . Les fonctions binomial_graph() et erdos_renyi_graph() sont des alias de cette fonction.

Paramètres :

n (int) - Le nombre de noeuds.

p (float) - Probabilité de création de l'arête.

seed (int, facultatif) - Amorce pour le générateur des nombres aléatoires (par défaut = Aucune).

directed (bool, facultatif (par défaut = False)) - Si la valeur est True, cette fonction renvoie un graphe orienté.

```
#importation de la biblioth que networkx
import networkx as nx

##importation de la biblioth que matplotlib pour affichage de graphe
import matplotlib.pyplot as plt

G= nx.erdos_renyi_graph(50,0.5)
nx.draw(G, with_labels=True)
plt.show()
```

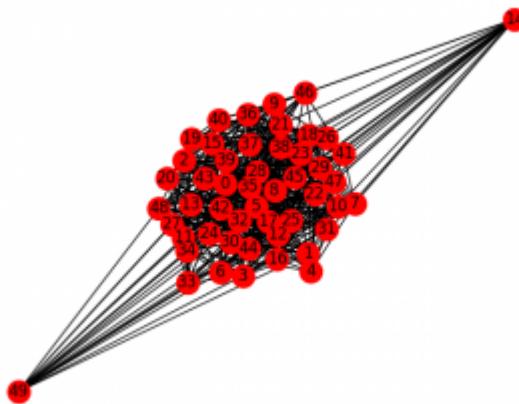


FIGURE 3.2.2 – Le résultat de la fonction "Erdos_renyi_graph(n,p,seed=None,directed=False)" Pour n=50,p=0.5

3.2.4 Complexité

Cet algorithme fonctionne dans $\theta(n^2)$, pour un graphe de densité faible (c'est-à-dire, pour des petites valeurs de p), fast_gnp_random_graph() est un algorithme plus rapide.

3.3 Modèle de Barabási-Albert

3.3.1 Définition générale

Le modèle de Barabási-Albert (BA) est un algorithme pour la génération aléatoire des réseaux sans échelle à l'aide d'un mécanisme d'attachement préférentiel. On pense que plusieurs systèmes naturels ou humains, tel que

l'Internet, le world wide web, les réseaux des citations, et certains réseaux sociaux sont approximativement sans échelle. Ils contiennent en tout cas quelques noeuds (appelés hubs ou moyeux) avec un degré inhabituellement élevé par rapport aux autres noeuds du réseau. Le modèle BA tente d'expliquer l'existence de tels noeuds dans de véritables réseaux. L'algorithme est nommé d'après ses inventeurs Albert-László Barabási et Réka Albert et pour un cas particulier d'un modèle plus général, il est appelé modèle de Price

3.3.2 L'avantage du modèle Barabási-Albert

Le modèle Barabási – Albert est l'un des nombreux modèles proposés générant des réseaux sans échelle. Il intègre deux concepts généraux importants : la croissance et l'attachement préférentiel. La croissance et l'attachement préférentiel existent largement dans les réseaux réels. La croissance signifie que le nombre des noeuds dans le réseau augmente avec le temps.

L'attachement préférentiel signifie que plus un noeud est connecté, plus il est susceptible de recevoir des nouveaux liens. Les noeuds avec un degré plus élevé ont une plus grande capacité à récupérer les liens ajoutés au réseau. Intuitivement, l'attachement préférentiel peut être compris si nous pensons en termes de réseaux sociaux reliant les personnes. Ici, un lien de A à B signifie que la personne A "sait" ou "connaît" la personne B. Les noeuds fortement liés représentent des personnes bien connues ayant beaucoup des relations. Lorsqu'un nouvel arrivant entre dans la communauté, il / elle est plus susceptible de faire la connaissance d'une de ces personnes plus visibles que d'un inconnu relatif. Le modèle BA a été proposé en supposant que, sur le World Wide Web, les nouvelles pages sont associées de préférence aux concentrateurs, c'est-à-dire aux sites très connus tels que Google, plutôt qu'à des pages que presque personne ne connaît. Si quelqu'un sélectionne une nouvelle page vers laquelle créer un lien en choisissant au hasard un lien existant, la probabilité de sélectionner une page particulière sera proportionnelle à son degré.

3.3.3 Implémentation du modèle Barabási-Albert

```
#importation de la biblioth que networkx
import networkx as nx
##importation de la biblioth que matplotlib pour affichage de graphe
import matplotlib.pyplot as plt
G= nx.barabasi_albert_graph(40,15)
nx.draw(G, with_labels=True)
plt.show()
```

Paramètres

-n : int : Nombre de noeuds

-m : int : Nombre d'arêtes à attacher d'un nouveau noeud à des noeuds existants

-seed : int, facultatif Seed pour le générateur de nombres aléatoires (défaut = Aucun)

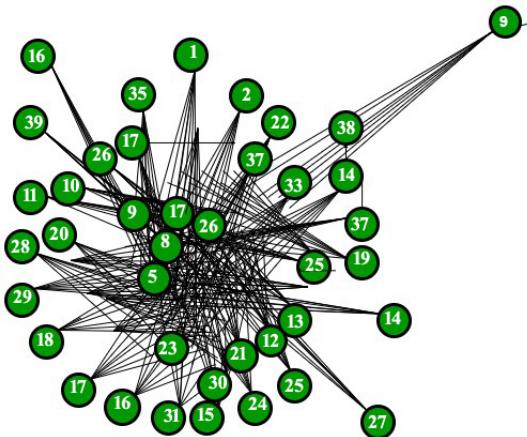


FIGURE 3.3-1 – Le résultat de la fonction "barabasi_albert_graph(40,15)" pour 40 noeuds

3.3.4 Complexité

Algorithme de Barabási-Albert (avec son implémentation modifiée et variante) devrait avoir une complexité temporelle moyenne de $\Theta(n^2)$ pour les réseaux en croissance de noeuds d'ordre n. Cela est dû au fait que chaque itération d'algorithme doit explorer tous les noeuds du réseau actuel en vue de l'insertion éventuelle des nouvelles liaisons.

3.4 Modèle de Watts-Strogatz

3.4.1 Définition générale

Le modèle watts – Strogatz est un modèle de génération des graphes aléatoires qui produit des graphes avec des propriétés de petit monde, y compris des longueurs de courtes chemin moyennes et un clustering élevé. Il a été proposé par Duncan J. watts et Steven Strogatz dans leur document conjoint 1998 nature.

3.4.2 L'avantage du modèle Watts-Strogatz

Cependant, les graphes ER n'ont pas deux propriétés importantes observées dans de nombreux réseaux du monde réel :

1. ils ne génèrent pas de regroupements locaux et de fermetures triadiques(clustering). Au lieu de cela parce qu'ils ont une probabilité constante, aléatoire et indépendante de deux noeuds étant connectés, les graphes ER ont un faible coefficient de clustering.
2. ils ne représentent pas la formation des hubs (points centraux). Formellement, le degré de distribution des graphes ER converge vers une distribution de Poisson, plutôt qu'une loi de puissance observée dans de nombreux réseaux réels et sans échelle.

Le modèle watts-Strogatz a été conçu comme le modèle le plus simple possible qui aborde la première des deux limitations. Il tient compte de l'agrégation tout en conservant les longueurs moyennes courtes de chemin du modèle ER. Il le fait en interpolant entre un graphe ER et un treillis circulaire régulier. Par conséquent, le

modèle est en mesure d'expliquer au moins partiellement les phénomènes du « petit monde » dans une variété de réseaux, tels que le réseau électrique, les réseaux neuronaux. et un réseau d'acteurs du cinéma.

3.4.3 Implémentation du modèle Watts-Strogatz

```
#importation de la bibliothque networkx
import networkx as nx
# importation de la bibliothque matplotlib pour affichage de graphe
import matplotlib.pyplot as plt
G= nx.watts_strogatz_graph(100,10,0.2)
nx.draw(G, with_labels=True)
plt.show()
```

Paramètres

-n : int : Nombre de noeuds

-m : int : Nombre de noeuds le plus proche dans la topologie cyclique à attacher d'un noeud

-p : float, la probabilité de attacher chaque arc

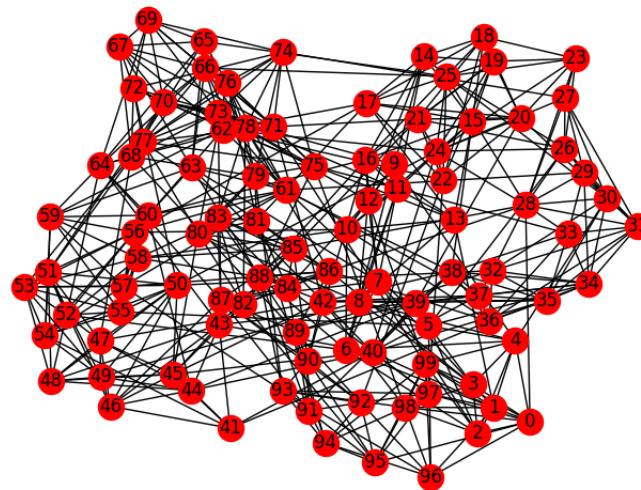


FIGURE 3.4-1 – Le résultat de la fonction "G= nx.watts_strogatz_graph(100,10,0.2)" pour 100 noeuds

3.4.4 Complexité

Pour le module de W-S la complexité varie entre $\theta(n^2 * m)$ et $\theta(n^2 * p * m)$.

Il est difficile de déterminer la complexité moyenne car W-S fonctionne sur deux phases :

la première est le création d'un graphe cyclique uniforme, la deuxième est la modification d'ensemble de arcs pour augmenter le nombre de triangles(coefficient de clustering).

4 Application

Dans ce chapitre nous introduisons notre propre travail qui consiste à extraire les graphes réels à partir de réseau de communication Twitter en utilisant Gephi puis nous les générerons par des graphes aléatoires (avec plusieurs modules) qui ont les mêmes tailles avec les réels et les comparer graphiquement et caractéristiquement.

4.1 Détection des communautés

4.1.1 Notre implementation

Dans notre algorithme, nous avons utilisé la proche d'optimisation de modularité avec Edge_Betweenness(E-B), la classe communauté a été définie initialement par le graphe entier pour obtenir les partitions les plus utiles on utilisant la notion de edge_betweenness qui décrit le nombre des plus courts chemins passant par un certain arc ,si un arc a une valeur importante de EB alors il est probablement un arc qui lie deux communautés donc par la suppression de cet arc nous avons des composants qui ont des degrés d'intérieur élevé par rapport à les arcs d'extérieur. Alors la modularité va agrandir.

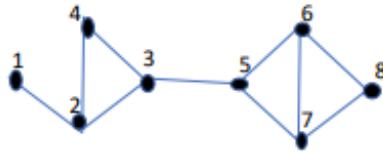
Le principe est de supprimer l'arc à la valeur maximal de E-B et calculer la modularité après cette itération s'il est améliorée on enregistre les changements sinon on garde l'arc.

On répète cette itération jusqu'à ce qu'on ne peut pas obtenir une valeur mieux, la valeur finale décrit la modularité maximale du graphe et par la suite la meilleure division en partitions

Pour réaliser cet implementaion nous avons utilisé les fonctions suivantes :

Nom de Fonction	Fonctionnalité	Les paramètre	Le retour
Find_best_partition	Permet de partitionner le graphe en sous-graphes par la suppression d'arcs qui ont le facteur E_B plus important.	• Self : les attributs de classe.	• Partitions :sous forme d'un dictionnaire de données G de type graphe. • Removed_edges :les arcs supprimés sous forme d'un tuple des données.
Get_maxbetweenness_edges	Permet de calculer le facteur de E_B (le nombre des plus courts chemins passant par un arc).	• Self : les attributs de classe. • Betweenness : les arcs qui nous voulons calculer.	• Max_betweenness_edges :les arcs qui ont le coefficient de E_B maximal.
Build_levels	Faire un procédure en largeur (BFS)	• Self : les attributs de classe. • G : le graphe. • Root : nœud de départ	• Levels : liste des nœuds qui sont distribués sur les niveaux • Predecessors • Successors (créer un arbre par le BFS)
Calculate_credits	Permet de Calculer le facteur E_B	• Self : les attributs de classe. • Levels : liste des nœuds qui sont distribués sur les niveaux • Predecessors • Successors	• Nodes_credit :les nombres de PCC passant par les nœuds. • Edges_credit :les nombres de PCC passant par les arcs.
My_betweenness_calculation	Permet de Retourner l'arc qui a la plus grande valeur de E_B.	• Self : les attributs de classe. • G : le graphe. • Normalized : booléen par default =false	Edge_contributions : L'arc qui a la plus grande valeur de E_B.

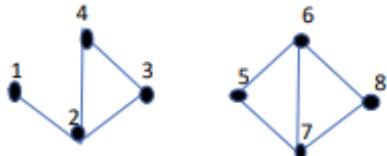
Notre implémentaion se fonctionne sous le principe suivant :



L'arc	Facteur de E_B
$1 \leftrightarrow 2$	14
$2 \leftrightarrow 3$	20
$2 \leftrightarrow 4$	10
$3 \leftrightarrow 4$	4
$3 \leftrightarrow 5$	32
$5 \leftrightarrow 6$	18
$5 \leftrightarrow 7$	12
$6 \leftrightarrow 7$	2
$6 \leftrightarrow 8$	4
$7 \leftrightarrow 8$	10

L'arc a le facteur E_B le plus grand est $3 \leftrightarrow 5$

Lorsque on supprime cet arc le graphe devient :



D'après la fonction de la modularité on trouve les deux composants $(1,2,3,4)$ et $(5,6,7,8)$ qui donnent la valeur maximale de Q par conséquence ces deux composants représentants les communautés de ce graphe.

On répète cette itération jusqu'à ce qu'on ne peut plus augmenter la modularité.

Les chemins les plus courts entre les nœuds :

1	2				
1	2	4			
1	2	3			
1	2	3	5		
1	2	3	5	6	
1	2	3	5	7	
1	2	3	5	6	8
2	3				
2	4				
2	3	5			
2	3	5	6		
2	3	5	7		
2	3	5	6	8	
3	4				
3	5				
3	5	6			
3	5	7			
3	5	7	8		
4	3	5			
4	3	5	6		
4	3	5	7		
4	3	5	6	8	
5	6				
5	7				
5	6	8			
6	7				
6	8				
7	8				

Voici code source :

```

from operator import mul
import networkx as nx
import os
import sys
import community
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import matplotlib.colors as mpcolors
import matplotlib.cm as mpcm
import numpy as np

#                               class Communautes
class Communities:
    def __init__(self, ipt_txt, ipt_png):
        self.ipt_txt = ipt_txt
        self.ipt_png = ipt_png
        self.graph = None

    def initialize(self):
        if not os.path.isfile(self.ipt_txt):
            self.quit(self.ipt_txt + " doesn't exist or it's not a file")
        # initialiser les bibliothèques
        self.graph = nx.Graph()
        # lire les données
        self.load_txt(self.ipt_txt)

    #                               Fonctions principales
    def find_best_partition(self):
        G = self.graph.copy()
        modularity = 0.0
        removed_edges = []
        partition = {}
        while 1:
            betweenness = self.calculte_betweenness(G)
            max_betweenness_edges = self.get_max_betweenness_edges(betweenness)
            if len(G.edges()) == len(max_betweenness_edges):
                break

            G.remove_edges_from(max_betweenness_edges)
            components = nx.connected_components(G)
            idx = 0
            tmp_partition = {}
            for component in components:
                for inner in list(component):
                    tmp_partition.setdefault(inner, idx)
                idx += 1
            cur_mod = community.modularity(tmp_partition, G)

            if cur_mod < modularity:
                G.add_edges_from(max_betweenness_edges)
                break;
            else:
                partition = tmp_partition
                removed_edges.extend(max_betweenness_edges)
                modularity = cur_mod
        return partition, G, removed_edges

```

```

def get_max_betweenness_edges(self, betweenness):
    max_betweenness_edges = []
    max_betweenness = max(betweenness.items(), key=lambda x: x[1])
    for (k, v) in betweenness.items():
        if v == max_betweenness[1]:
            max_betweenness_edges.append(k)
    return max_betweenness_edges

def calcule_betweenness(self, G, bonus=True):
    """Calculer Betweenness
    entree:
    - G: graphique
    - Bonus: True si utiliser ma propre calculatrice betweenness. (Bonus = True par defaut)
    """
    if bonus:
        betweenness = self.my_betweenness_calculation(G)
    else:
        betweenness = nx.edge_betweenness_centrality(G, k=None, normalized=True,
                                                    weight=None, seed=None)
    return betweenness

# Fonctions auxiliaire
def build_level(self, G, root):
    """
    Niveau pour le graphe
    entree:
    - G: networkx graph
    - racine : noeud racine
    sortie :
    - niveaux: noeuds dans chaque niveau
    - predecesseurs : predecesseurs pour chaque noeud
    - successeurs : successeurs pour chaque noeud
    """
    levels = {}
    predecessors = {}
    successors = {}

    cur_level_nodes = [root] # initialize le point de depart
    nodes = [] # stocke les noeuds qui ont ete consultes
    level_idx = 0
    while cur_level_nodes: # si ont des noeuds pour un niveau, continuer le processus
        nodes.extend(cur_level_nodes) # ajouter des noeuds qui sont l'interieur de nouveau niveau dans la liste des noeuds
        levels.setdefault(level_idx, cur_level_nodes) # definir des noeuds pour niveau actuel
        next_level_nodes = [] # preparer des noeuds pour les niveaux suivants

        # trouver noeud dans la prochaine etape
        for node in cur_level_nodes:
            nei_nodes = G.neighbors(node) # tous les voisins pour le noeud dans le niveau actuel
            # trouver des noeuds voisins dans le niveau suivant
            for nei_node in nei_nodes:

```

```

        if nei_node not in nodes:      # noeuds dans le niveau suivant ne doit
                                        # pas etre accede
            predecessors.setdefault(nei_node, [])    # initialiser le
                                                # dictionnaire
                                                # predecesseurs,
                                                # utiliser une
                                                # liste pour
                                                # stocker tous
                                                # les
                                                # predecesseurs
            predecessors[nei_node].append(node)
            successors.setdefault(node, [])      # initialiser le
                                                # dictionnaire
                                                # successeurs,
                                                # utiliser une
                                                # liste pour
                                                # stocker tous
                                                # les successeurs
            successors[node].append(nei_node)

        if nei_node not in next_level_nodes:    # eviter d'ajouter meme
                                                # noeud deux
                                                # fois
            next_level_nodes.append(nei_node)
        cur_level_nodes = next_level_nodes
        level_idx += 1
    return levels, predecessors, successors

def calculate_credits(self, G, levels, predecessors, successors, nodes_nsp):
    """
    Calculer les credits pour les noeuds et les bords
    """
    nodes_credit = {}
    edges_credit = {}
    # boucle, de bas en haut, sans inclure le niveau zero
    for lvl_idx in range(len(levels)-1, 0, -1):
        lvl_nodes = levels[lvl_idx]      # obtenir des noeuds dans le niveau
        # calculer pour chaque noeud du niveau actuel
        for lvl_node in lvl_nodes:
            nodes_credit.setdefault(lvl_node, 1.)    # definir le credit par defaut
                                                # pour le noeud, 1
            if successors.has_key(lvl_node):          # si ce n'est pas un noeud
                                                # feuille
                # Chaque noeud qui n'est pas une feuille obtient le credit = 1 +
                                                # somme des credits
                                                # des bords de ce
                                                # noeud au niveau ci-
                                                # dessous
                for successor in successors[lvl_node]:
                    nodes_credit[lvl_node] += edges_credit[(successor, lvl_node)]
            node_predecessors = predecessors[lvl_node]    # obtenir des
                                                # predecesseurs du noeud
                                                # dans le niveau actuel
            total_nodes_nsp = .0      # nombre total de chemins plus courts pour les
                                                # predecesseurs du noeud
                                                # au niveau actuel

```

```

        for predecessor in node_predecessors:
            total_nodes_nsp += nodes_nsp[predecessor]

        for predecessor in node_predecessors:
            predecessor_weight = nodes_nsp[predecessor]/total_nodes_nsp
            edges_credit.setdefault((lvl_node, predecessor), nodes_credit[
                lvl_node]*predecessor_weight)

    return nodes_credit, edges_credit

def my_betweenness_calculation(self, G, normalized=False):
    # Fonction de calcul du betweenness
    graph_nodes = G.nodes()
    edge_contributions = {}
    components = list(nx.connected_components(G))      # composants connectés pour
                                                       # graph en cours

    # calculer pour chaque noeud
    for node in graph_nodes:
        component = None      # le courant communautaire auquel appartient le noeud
        for com in components:
            if node in list(com):
                component = list(com)
        nodes_nsp = {} # nombre de "shortest paths"
        node_levels, predecessors, successors = self.build_level(G, node)  # construction des niveaux
                                                                           # pour les calculs
        # calculer des chemins plus courts pour chaque noeud (y compris le noeud
        # actuel)
        for other_node in component:
            shortest_paths = nx.all_shortest_paths(G, source=node, target=other_node)
            nodes_nsp[other_node] = len(list(shortest_paths))
        # calculer des credits pour les noeuds et les bords (Utilisez uniquement "
        # edges_credit" en fait)
        nodes_credit, edges_credit = self.calculate_credits(G, node_levels,
                                                               predecessors, successors,
                                                               nodes_nsp)

        # triple de tri (valeur cle de edges_credit), et resumer pour
        # edge_contributions
        for (k, v) in edges_credit.items():
            k = sorted(k, reverse=False)
            edge_contributions_key = (k[0], k[1])
            edge_contributions.setdefault(edge_contributions_key, 0)
            edge_contributions[edge_contributions_key] += v

        # diviser par 2 pour obtenir la vraie betweenness
        for (k, v) in edge_contributions.items():
            edge_contributions[k] = v/2

        # Normaliser
        if normalized:
            max_edge_contribution = max(edge_contributions.values())
            for (k, v) in edge_contributions.items():
                edge_contributions[k] = v/max_edge_contribution
    return edge_contributions

```

```
#                                     Methode Main
if __name__ == '__main__':
    ipt_txt = sys.argv[1]
    ipt_png = sys.argv[2]

    c = Communities(ipt_txt, ipt_png)
    c.initialize()
    partition, part_graph, removed_edges = c.find_best_partition()
    c.display(partition)
    c.plot_graph(part_graph, removed_edges)
```

4.2 Générer un graphe réel

Notre travail consiste à extraire des graphes réels à partir d'un réseau social.

Sur Twitter on choisit deux sujets le premier est le UCL (The UEFA Champions League) est une organisation de football compétition dans l'European Union Association de football (UEFA).

Dans ce sujet nous avons extrait quatre graphes avec des dimensions croissantes pour montrer l'évolution d'un réseau de communication.

Le deuxième est un sujet mondial (fête mondiale des femmes "womensday") qui a donné un graphe de grande taille, ce graphe présente les caractéristiques des réseaux complexes.

I ucl100

- nombre des noeuds = 143
- nombre des arcs = 167

II ucl500

- nombre des noeuds = 504
- nombre des arcs = 749

III ucl1000

- nombre des noeuds = 1001
- nombre des arcs = 1493

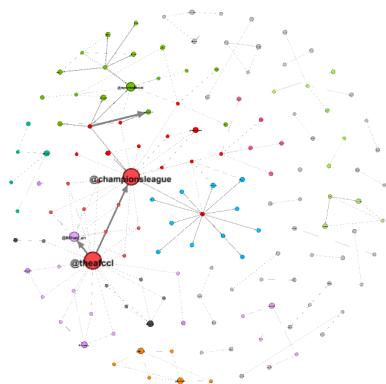
IV ucl4000

- nombre des noeuds = 4004
- nombre des arcs = 6246

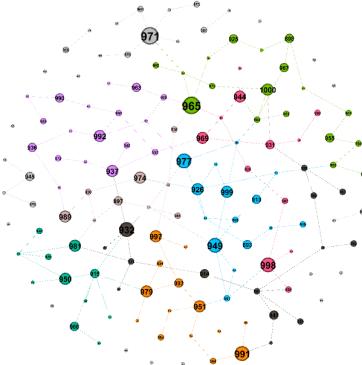
V womensday20000

- nombre des noeuds = 20167
- nombre des arcs = 42852

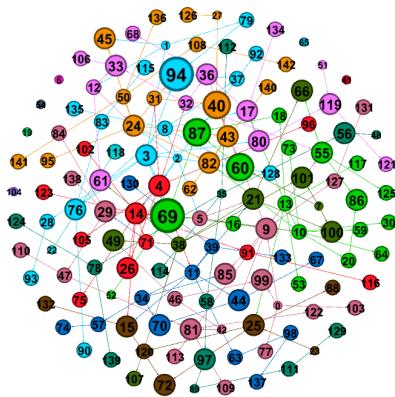
4.2.1 UCL 100



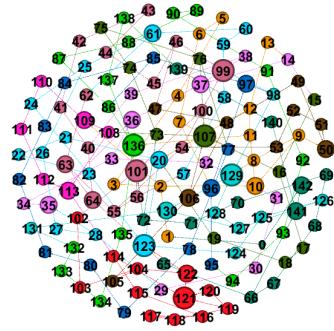
(a) graphe réel



(b) graphe d'Erdos-renyi



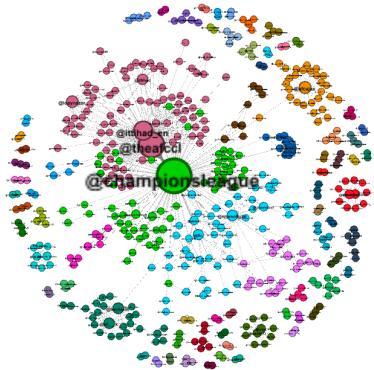
(c) graphe de barabasi



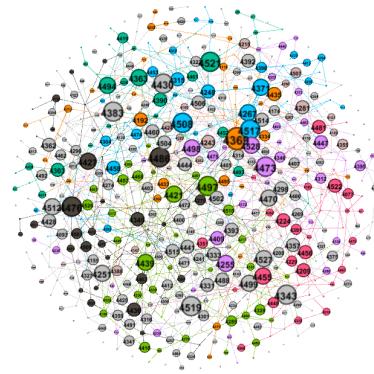
(d) graphe de watts-strogatz

FIGURE 4.2-1 – les graphes de UCL 100

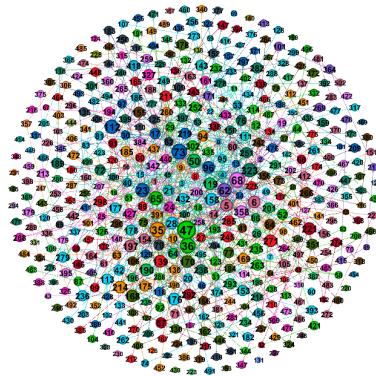
4.2.2 UCL 500



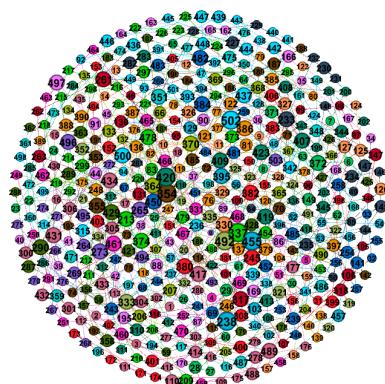
(a) graphe réel



(b) graphe d'Erdos-renyi



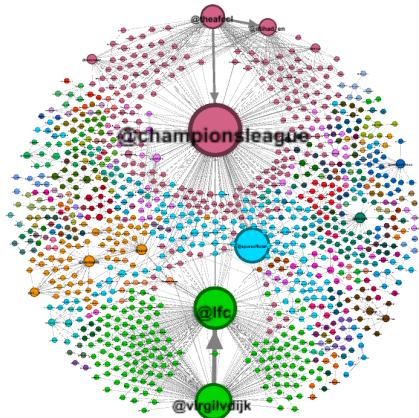
(c) graphe de barabasi



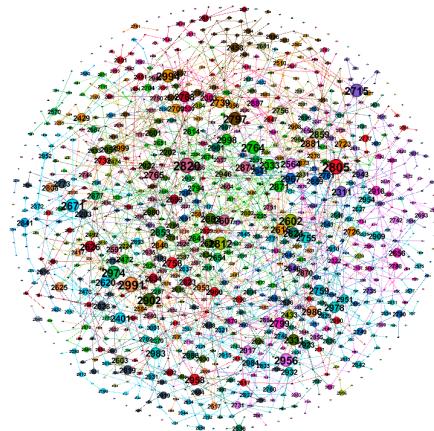
(d) graphe de watts-strogatz

FIGURE 4.2-2 – les graphes de UCL 500

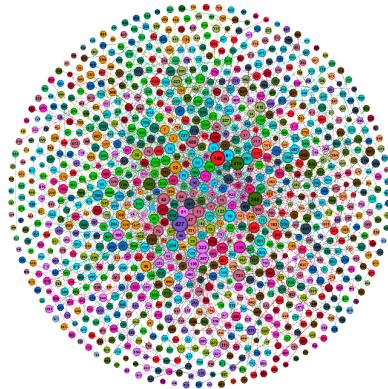
4.2.3 UCL 1000



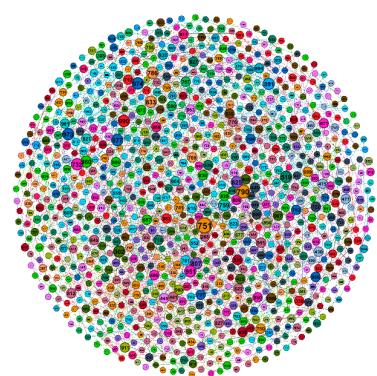
(a) graphe réel



(b) graphe d'Erdos-renyi



(c) graphe de barabasi



(d) graphe de watts-strogatz

FIGURE 4.2-3 – les graphes de UCL 1000

4.2.4 UCL 4000

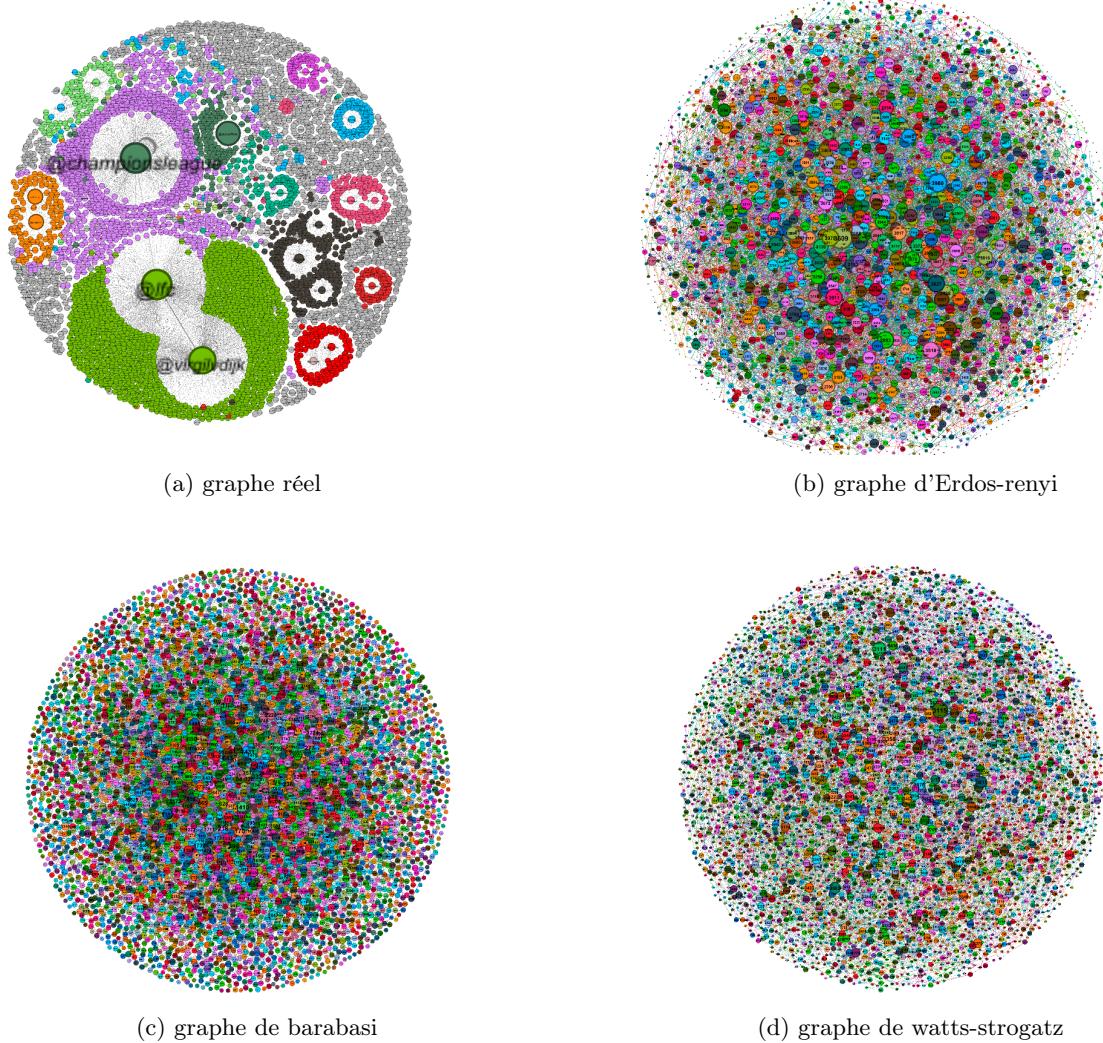


FIGURE 4.2-4 – les graphes de UCL 4000

On remarque graphiquement que les graphes aléatoires ne peuvent pas arriver à représenter le réseau réel avec ses propriétés des communautés même avec plusieurs modules.

4.3 Comparaison des caractéristiques

4.3.1 Module de Erdos-renyi base sur les graphes de terrain :

	Type de graph	Ucl 150		Ucl 500		Ucl 1000		Ucl 4000		WD 20000	
Nombre des nœuds	Réel	143	--	504	--	1001	--	4004	--	20167	--
	Aléatoire	143	--	504	--	1001	--	4004	--	20167	--
Nombre Des Arrêts	Réel	167	--	749	--	1493	--	6246	--	42851	--
	Aléatoire	142	--	733	--	1510	--	6477	--	40698	--
Degré Moyenne	Réel	1.168	15%	1.486	2%	1.492	1%	1.56	4%	2.125	5%
	Aléatoire	0.993		1.454		1.508		1.618		2.018	
Diamètre	Réel	3	33%	4	75%	5	100%	5	80%	5	180%
	Aléatoire	4		7		10		9		14	
Densité	Réel	0.008	4%	0.003	0%	0.0015	0%	0.00039	3%	0.0001	0%
	Aléatoire	0.007		0.003		0.0015		0.00037		0.0001	
Modularité	Réel	0.747	5%	0.710	19%	0.750	16%	0.722	17%	0.811	38%
	Aléatoire	0.712		0.575		0.632		0.604		0.505	
Cluster	Réel	1.093	100%	0.128	98%	0.25	99%	0.321	100%	0.075	100%
	Aléatoire	0		0.003		0.002		0.00001		0	
Plus court chemin moyen	Réel	1.454	4%	1.865	25%	2.227	24%	2.338	14%	1.868	82%
	Aléatoire	1,512		2,324		2,757		2.665		3.384	
Nombre des communautés	Réel	30	3%	76	44%	136	37%	300	23%	??	
	Aléatoire	29		42		85		233		??	

4.3.2 Module de Barabasi-albert base sur les graphes de terrain :

	Type de graph	Ucl 150		Ucl 500		Ucl 1000		Ucl 4000		WD 20000	
Nombre des nœuds	Réel	143	--	504	--	1001	--	4004	--	20167	--
	Aléatoire	143	--	504	--	1001	--	4004	--	??	--
Nombre Des Arrêts	Réel	167	--	749	--	1493	--	6246	--	42851	--
	Aléatoire	142	--	733	--	1510	--	6477	--	??	--
Degré Moyenne	Réel	1.168	1.9%	1.486	0.8%	1.492	1.4%	1.56	1.7%	2,125	--
	Aléatoire	1.189		1.498		1.47		1.588		??	--
Diamètre	Réel	3	133%	4	100%	5	100%	5	200%	5	--
	Aléatoire	7		8		10		15		??	--
Densité	Réel	0.008	0%	0.003	0%	0.0015	3%	0.00039	3%	0.0001	--
	Aléatoire	0.008		0.003		0.001		0.00040		??	--
Modularité	Réel	0.747	7%	0.710	14%	0.750	15.4%	0.722	15.4%	0.811	--
	Aléatoire	0.681		0.608		0.634		0.611		??	--
Cluster	Réel	1.093	98%	0.128	78%	0.25	92%	0.321	93%	0.075	--
	Aléatoire	0.0170		0.0304		0.0166		0.0053		??	--
Plus court chemin moyen	Réel	1.454	78%	1.865	123.3%	2.227	49%	2.338	85%	1.868	--
	Aléatoire	2.528		3.132		3.322		4.377		??	--
Nombre des communautés	Réel	30	43%	76	40%	136	44.8%	300	29%	??	--
	Aléatoire	17		45		75		213		??	--

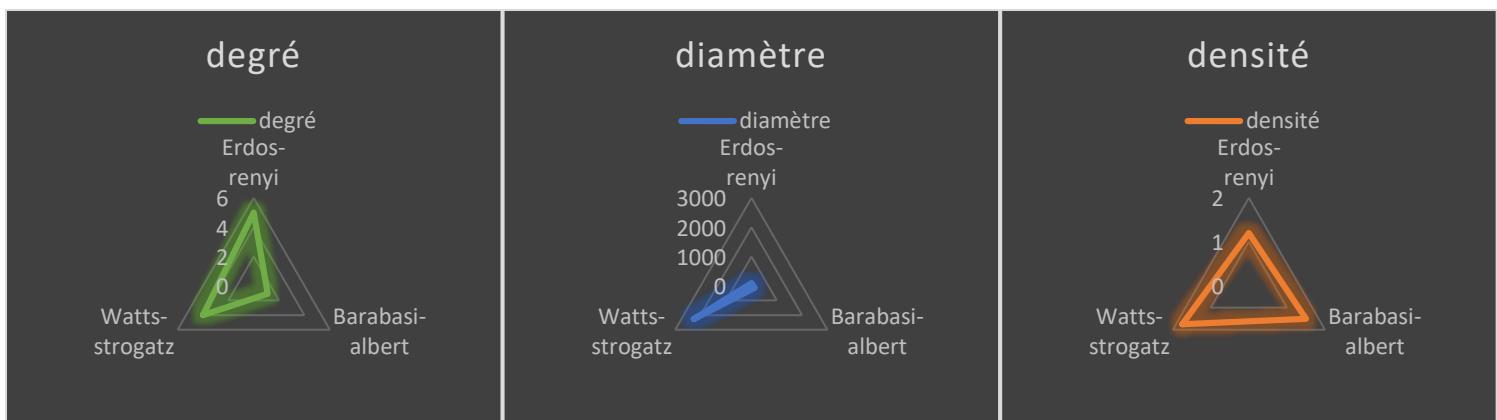
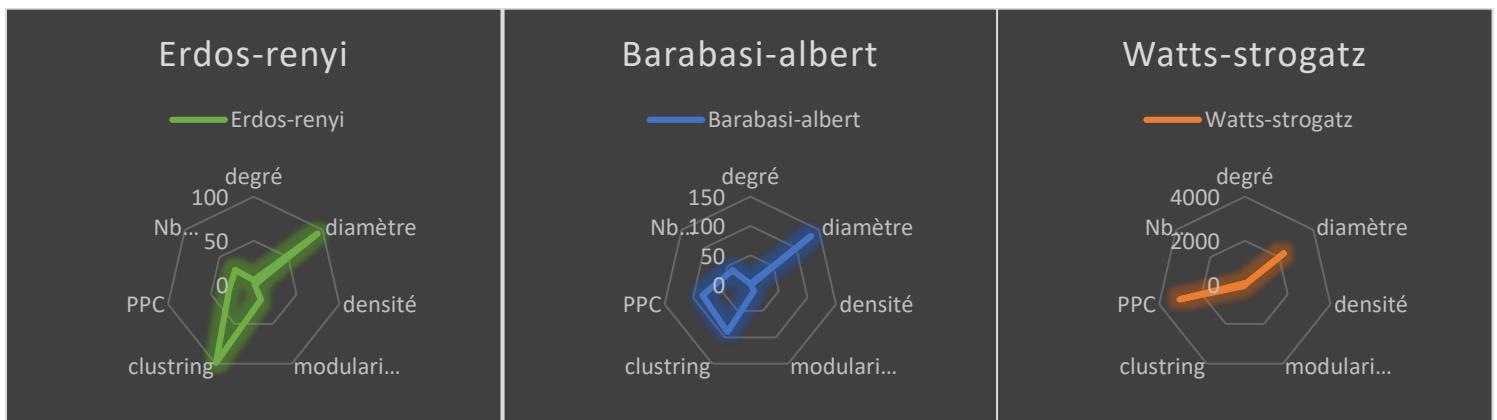
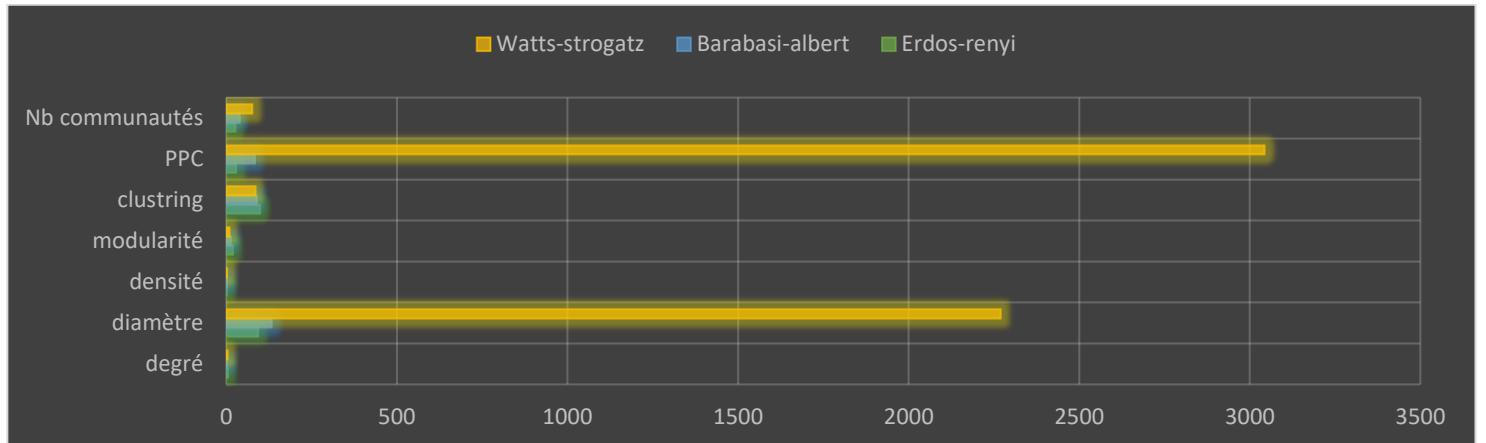
4.3.3 Module de Watts-strogatz base sur les graphes de terrain :

		Type de graph	Ucl 150		Ucl 500		Ucl 1000		Ucl 4000		WD 20000	
Nombre des nœuds	Réel	143	--	504	--	1001	--	4004	--	20167	--	
	Aléatoire	143		504		1001		4004		??		
Nombre Des Arrêts	Réel	167	--	749	--	1493	--	6246	--	42851	--	
	Aléatoire	150		743		1550		6377		??		
Degré Moyenne	Réel	1.168	10%	1.486	2%	1.492	1%	1,56	3%	2,125	??	
	Aléatoire	1.217		1476		1.486		1.603		??		
Diamètre	Réel	3	1500%	4	2100%	5	1680%	5	3800%	5	??	
	Aléatoire	48		88		89		195		??		
Densité	Réel	0.008	2%	0.003	0%	0.0015	2%	0.00039	3%	0.0001	??	
	Aléatoire	0.009		0.003		0.001		0.0004		??		
Modularité	Réel	0.747	4.1%	0.710	8%	0.750	12.4%	0.722	13.8%	0.811	??	
	Aléatoire	0.716		0.653		0.657		0.627		??		
Cluster	Réel	1.093	90%	0.128	80%	0.25	85%	0.321	90%	0.075	??	
	Aléatoire	0.010		0.0036		0.0036		0.0036		??		
Plus court chemin moyen	Réel	1.454	1100%	1.865	8888%	2.227	863%	2.338	1350%	1.868	??	
	Aléatoire	15.956		17.517		21.006		29.683		??		
Nombre des communautés	Réel	30	60%	76	73%	136	83%	300	88%	??	??	
	Aléatoire	12		20		23		36		??		

19

19. Remarque :le graphe womensday est trop grand pour le générer c'est pour cela nous avons rencontrés des contraintes d'ordre technique(serveur) qui ne nous a pas permis d'aboutir au résultat

Nous comparons les résultats des tableaux ci-dessus afin de donner les graphiques suivants qui représentent l'erreur des trois modules par rapport à les graphes réels



Conclusion et Perspectives

Notre travail a porté sur comment extraire les graphes dans un réseau social nous avons utilisé Twitter , l'analyse des ces graphes fait par utilisation de igraph et networkx comme des bibliothèque de python et Gephi sur la plate-forme Java, puis on les générer par des graphes aléatoires au but de comparer leurs caractéristiques avec les réels.

Tout pour avoir une vue générale sur un réseau social et de son évolution avec le temps. nous essayons d'utiliser différents modules pour obtenir la meilleure adéquation avec la structure du réseau social. Il en résulte que les modèles aléatoires ne peuvent pas obtenir tous les arguments nécessaires pour obtenir la version exacte ou au moins une version proche du réseau réel. mais nous obtenons un résultat intéressant en changeant les modules ,On a fait trois modules le premier c'est Erdos-Renyi) qui utilise uniquement la probabilité **p** de l'existence d'un arc entre deux sommets, a chaque fois on ajoute un nouvel sommet **i** E-R utiliser la fonction de probabilité base sur la valeur de p pour le choix de la création des arcs entre i et les sommets déjà créées. Le deuxième module c'est Albert-Barabasi qui utiliser la dégrée **d** si on ajoute un sommet **j** B-A créer d arcs entre j et les sommets existants, a ce moment B-A est statique pour renforcer la notion aléatoire nous avons utilisé la version entendue qui prend un paramètre de plus, ce paramètre est le probabilité de la recréation d'un sommet ou l'ajouter un nouvel. Le dernier module c'est Watts-Strogatz qui crée un ensemble des sommet sous une forme cyclique et on donne le paramètre **a** et W-S va créer des arcs avec les a sommets les plus proches puis recréer ces arcs selon une probabilité passe en paramètres.

Nous avons essayé d'obtenir un modèle qui intègre la structure de la communauté dans le graphe aléatoire ,nous avons eu un problème majeur de la mémoire lorsque on utilise l'algorithme sur le grand graphe (womensday), nous avons implémenté un algorithme qui détecte les communautés sous le terme de modularité de G-N mais avec une modification dans la fonctionnalité, G-N utiliser une approche d'agglomération comme nous avons mentionné dans le chapitre 2.3.1, dans notre implémentaion nous avons utilise l'approche séparative et améliorer on utilisant le facteur E-B pour donner l'efficacité au choix des arcs a supprimer (le principe et la fonctionnalité sont dans le chapitre 4.1.1).

Enfin, ce projet était une opportunité pour bien apprendre la théorie des graphes et les graphes aléatoires et les techniques de programmation comme les bibliothèques de python et de maitriser l'utilisation de logiciel GEPHI

le plus connue au niveau de l'analyse des donnée.

Nous présentons les modèles les plus populaires de graphiques introduites pour décrire des systèmes réels, au moins dans une certaine mesure. Ces graphiques sont des modèles nuls utiles pour la détection des communautés, car ils ne possèdent pas la structure de la communauté, ils peuvent donc être utilisés pour des tests négatifs de clustering. algorithmes.

Pour le moment, aucun modèle de ce type ne peut générer un bon graphe aléatoire pour un réseau réel, mais il y a une énorme amélioration dans ce domaine. Nous nous attendons à ce qu'un modèle proche puisse générer un réseau avec toutes les communautés. et les carasterstiques du réseau complexe

Bibliographie

- [02] GIRVAN M., NEWMAN M. E. J., « Community structure in social and biological networks », Proceedings of the National Academy of Sciences of the United States of America, vol. 99, *n°* 12, 2002, p. 7821–7826.
- [05] SCHAEFFER S. E., « Graph clustering », Computer Science Review, vol. 1, *n°* 1, 2007, p. 27–64.
- [10] FORTUNATO S., « Community detection in graphs », Physics Reports, vol. 486, *n°* 3-5, 2010.
- [12] PALLA G., DERENYI I., FARKAS I., VICSEK T., « Uncovering the overlapping community structure of complex networks in nature and society », Nature, vol. 435, *n°* 7043, 2005, p. 814–818
- [14] PONS P., « Détection de communautés dans les grands graphes de terrain », PhD thesis, 2007.
- [08] BLONDEL V. D., GUILLAUME J., LAMBIOTTE R., LEFEBVRE E., « Fast unfolding of communities in large networks », Journal of Statistical Mechanics : Theory and Experiment, vol. 10, 2008.
- [09] ROSVALL M., BERGSTROM C. T., « Maps of random walks on complex networks reveal community structure », Proceedings of the National Academy of Sciences, vol. 105, *n°* 4, 2008, p. 1118–1123.
- [09] CLAUSET A., NEWMAN M. E. J., MOORE C., « Finding community structure in very large networks », Phys. Rev. E, vol. 70, *n°* 6, 2004.
- [17] GAUME B., « Balades aléatoires dans les petits mondes lexicaux », I3 Information Interaction Intelligence, vol. 4, *n°* 2, 2004.
- [27] K. Borner, S. Sanyal et A. Vespignani. Network Science. Blaise Cronin (Ed) Annual Review of Informa-

tion Science and Technology, vol. 41, pages 537–607, 2007. (Cit  en pages 2, 14, 16, 22, 23 et 62.)

- [3] Uri Alon. Network motifs : theory and experimental approaches. *Nature Reviews Genetics*, 8 :450461, 2007.

<https://github.com/networkx>

- [4] M. S. Aldenderfer and R. K. Blashfield. Cluster Analysis. Number 07-044 in Sage University Paper Series on Quantitative Applications in the Social Sciences. Sage, Beverly Hills, 1984.

- [6] David Auber, Yves Chiricota, Fabien Jourdan, and Guy Melan on. Multiscale visualization of small world networks. In Proceedings of the 9th IEEE Symposium on Information Visualization (InfoVis 2003), page 10, Seattle, USA, 2003. IEEE Computer Society.

- [7] James P. Bagrow and Erik M. Boltt. Local method for detecting communities. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 72(4) :046108, 2005

- [15] Aaron Clauset. Finding local community structure in networks. *Physical Review E*, 72 :026132, 2005.

- [16] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical Review E*, 70(6) :066111, 2004.

- [18] Luciano da Fontoura Costa. Hub-based community finding. arXiv :cond-mat/0405022, 2004.

- [19] L. Donetti and M. A. Mu oz. Detecting network communities : a new systematic and efficient algorithm. *Journal of Statistical Mechanics*, 2004(10) :10012, 2004

- [20] L. Donetti and M. A. Mu oz. Improved spectral algorithm for the detection of network communities. In

Modeling cooperative behavior in the social sciences, volume 779, pages 104107, 2005.

[25] B. S. Everitt, S. Landau, and M. Leese. Cluster Analysis. Hodder Arnold, London, 4th edition, 2001.

[39] David Harel and Yehuda Koren. On clustering using random walks. In Proceedings of the 21st Foundations of Software Technology and Theoretical Computer Science (FSTTCS'01), LNCS 2245, pages 1841, 2001.

[47] R. Kannan, S. Vempala, and A. Vetta. On clusterings : good, bad and spectral. In Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS'00), page 367, Washington, DC, USA, 2000. IEEE Computer Society.

[58] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69(6) :066133, 2004.

[68] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Dening and identifying communities in networks. *PNAS*, 101(9) :26582663, 2004.

[79] Stijn van Dongen. Graph Clustering by Flow Simulation. PhD thesis, University of Utrecht, May 2000.

[85] Fang Wu and Bernardo A. Huberman. Finding communities in linear time : A physics approach. *The European Physical Journal B*, 38 :331338, 2004.

[86] Luh Yen, Fabien Wouters, François Fouss, Michel Verleysen, and Marco Saerens. Clustering using a random walk based distance measure. In Proceedings of the 13th Symposium on Articial Neural Networks (ESANN 2005), pages 317324, 2005.

[88] Haijun Zhou and Reinhard Lipowsky. Network Brownian motion : A new method to measure vertex-

vertex proximity and to identify communities and subcommunities. In International Conference on Computational Science, pages 10621069, 2004.